Unpacking the non-unpackable

@unixfreaxjp - r2con2018

malwaremustdie.org

Introduction

1. A security folk by day
   - Incident Response malware incident analyst support at a cyber emergency center in Japan
   - Advanced cyber threat research team member
   - FIRST.org member of SOC team
2. My community give-back:
   - Analysis write up in English (MMD blog +etc) & JP language
   - Moderator of a subreddit on Linux malware research
   - Lecturer for Linux malware on ICS threat in national events
3. Activity in radare2:
   - Introducing r2 in malware analysis & IR on national *& world-wide*
   - With help from pancake & national community we formed r2jp
   - I support mainstream to do tests, bug reports, ideas
   - The only lecturer teaches radare2 in All Japan Security Camp

Some facts you should know beforehand..

1. My background: Open Source believer, UNIX OS fanboy, FreeBSD user, UNIX Software debugger & Reverse Engineering
2. I may block you in twitter (but maybe my script does it, not me..)
   If you're the one, poke me here & let's see what we can do about that.
3. maijin, trufae, xvilka helps me a lot to catch up with recent radare..
4. The greatest decision in my life was:
   When I decided to run # *make uninstall* at radare2 dir in /usr/ports/ (as FreeBSD user) and switch to r2 Github version..
5. First r2 folks I met are:
   @zlowram, @trufae then @xvilka
6. Yes I also never find @trufae sleeps!

I never had chance to poke back to Paul,
for his kind greets, pls allow me to poke him back..

Poke back to Paul, thanks & please put more screenshot of r2

Re:

My r2con2018 presentation highlights

1. My talk is about Linux binary packers and will explain how I handle them with my very basic radare2 skill.
2. I will try to share a practical methods in analyzing ITW packers, with approach from simple concept to complex one.
3. All methods are presented with screenshots and hopefully to finish within 30 minutes.. so I will be quick (sorry!) ping me afer for Q/A
4. I don't explain about:
   a. Windows OS binaries or other RE tool
   b. What is packers and how to basically (un)packed a binary
   c. How to spot the packed binary, yada yada..
5. Some warning about my presentation & my research
   a. I leaned radare myself via /usr/ports & helped GREATLY by r2 team to use r2!
   b. Mostly busy on handling incidents, so many stuff I want to write.. so little time
   c. If you can't see my twitter, you may didn’t see references I wrote in here before, it’s why I recaps them all now.. in the first part of this talk as the base.

Contents

1. Appetizer: Review on vanilla UPX packers, dissection & summary of several known packers I had researched.
   (this part contains basic info & a nice kickstart)
2. Some soup: Interesting ELF packers today (a good take-away for you)
3. Main course: Unknown packer I spotted ITW (non-unpackable one)

The concept of ELF unpacking will be shown in appetizer session, which will be used also in main course later, while before and on the “soup” session is adding you knowledge of various famous ELF packers ever seen and currently existed ones.

In the main course part I will show the dissection of a sample or two, (my) way to unpack it, and the alleged hidden motivation of why the original binaries are packed in this way...

Motto of the day

Q: I don't like ELF packers!
A: So do I, and no one does too, but…

“ FIRE UP YOUR R2 AND - EVERYTHING WILL BE OKAY! :) “

Part one - the appetizer

“ Dealing with the Vanilla UPX &
details of other ELF packers I researched
so far..“

Previous shares on ELF packers dissection

1. How I statically dealt with the custom UPX (aka Vanilla UPX)
2. ..and dynamic analysis on the above packers
3. ..and several ways on memory depacking (generally applied to all)

These has been shared either via my Twitter, YouTube or in Imgur (for them who can access it, due to OPSEC). BUT.. don't worry, let me sum them up on these first opening slides as appetizer..

....let's summing them up within one minute (...joke)

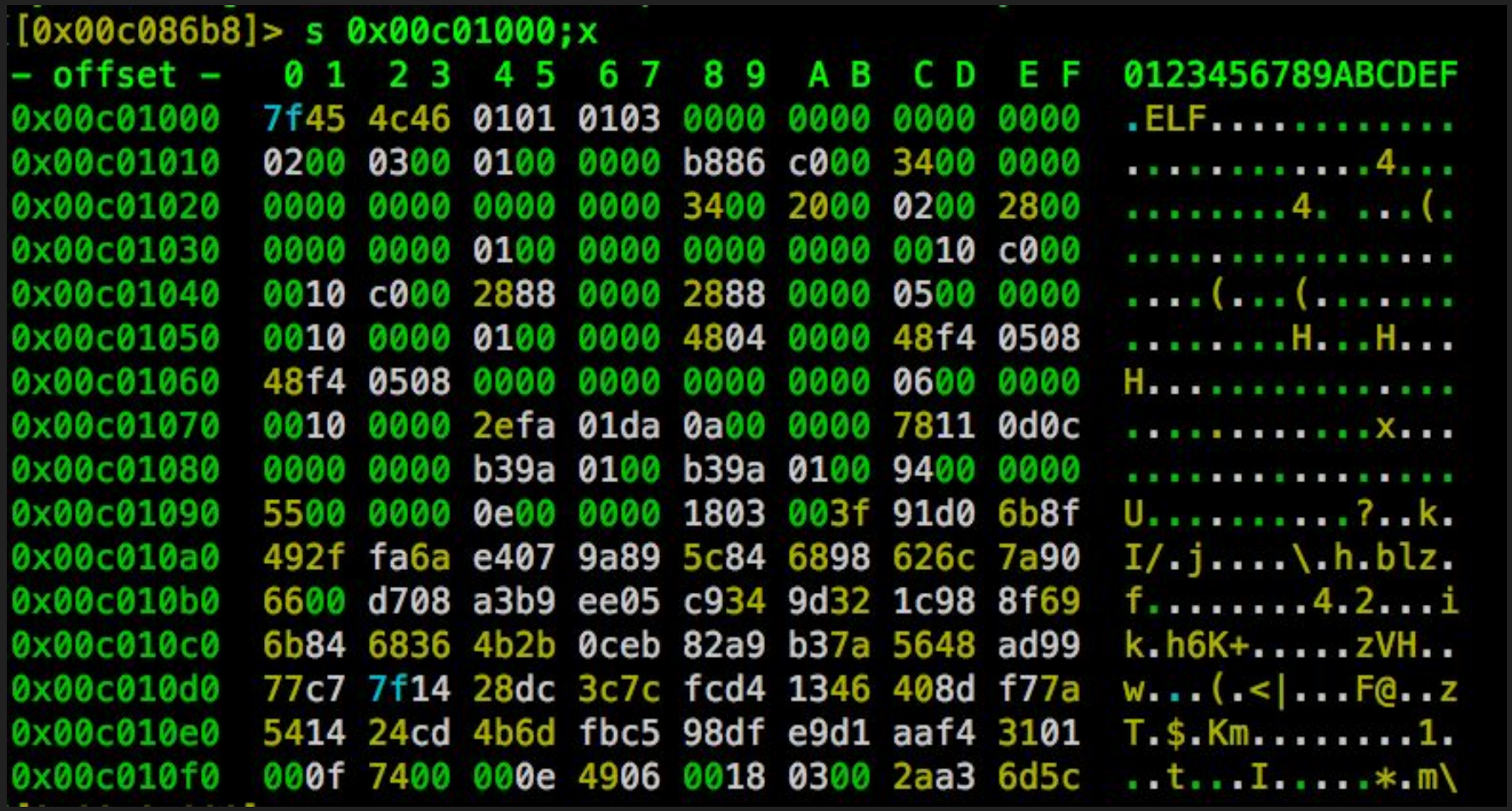
Vanilla ELF UPX 101 - in points & solution summary

1. Vanilla UPX 1st case - w/ UPX header rewriting method
   points are:
   - preventing the depacker command, the “-d”
   - making us believe this is not a UPX
   - they have no demerit by abusing headers, as long it runs well

   solution: Fixing the hex
2. Vanilla UPX 2nd case - w/ Source Code modification
   points are:
   - UPX is open source, we are all can see & can edit source code
   - Adding more "stuff" to UPX original packing (more routines etc)

   solution: Reading code then debugging it to modified code..
3. Other un-packing method if both two above doesn't work:
   - use radare2 to debug it,or brute it, or curse on it,
     and good luck!

...because you will maybe see this ELF



Anyone can tell what's wrong with this ELF?

The ELF header looks okay? (radare2)

```
sh-3.2$
sh-3.2$ r2 -nn sample
Module version mismatch /Users/rik/.local/share/radare2/plugins/core_pdd.dylib
 -- Don't feed the bugs! (except delicious stacktraces)!
[0x00000000]> pf.elf_header
      ident : 0x00000000 = .ELF....
       type : 0x00000010 = type (enum elf_type) = 0x2 ; ET_EXEC
    machine : 0x00000012 = machine (enum elf_machine) = 0x3 ; EM_386
    version : 0x00000014 = 0x00000001
      entry : 0x00000018 = 0x00c086b8
      phoff : 0x0000001c = 0x00000034
      shoff : 0x00000020 = 0x00000000
      flags : 0x00000024 = 0x00000000
     ehsize : 0x00000028 = 0x0034
  phentsize : 0x0000002a = 0x0020
      phnum : 0x0000002c = 0x0002
  shentsize : 0x0000002e = 0x0028
      shnum : 0x00000030 = 0x0000
   shstrndx : 0x00000032 = 0x0000
[0x00000000]>
```

Run: r2 -nn <sample> ; pff.elf_header

Check the header's values.. (elfdump)

```
e_ident: ELFCLASS32 ELFDATA2LSB ELFOSABI_LINUX
e_type: ET_EXEC
e_machine: EM_386
e_version: EV_CURRENT
e_entry: 0xc086b8
e_phoff: 52
e_shoff: 0
e_flags: 0
e_ehsize: 52
e_phentsize: 32
e_phnum: 2
e_shentsize: 40
e_shnum: 0
e_shstrndx: 0
```

Use: BSD command elfdump to dump the ELF header

How about the program header entries? (radare2)

```
[[0x00000000]>
[[0x00000000]>
[[0x00000000]> s 0x34
[[0x00000034]> pf.elf_phdr
    type : 0x00000034 = type (enum elf_p_type) = 0x1 ; PT_LOAD
  offset : 0x00000038 = 0x00000000
   vaddr : 0x0000003c = 0x00c01000
   paddr : 0x00000040 = 0x00c01000
  filesz : 0x00000044 = 0x00008828
   memsz : 0x00000048 = 0x00008828
   flags : 0x0000004c = flags (enum elf_p_flags) = 0x5 ; PF_Read_Exec
   align : 0x00000050 = 0x00001000
[[0x00000034]>
[[0x00000034]> s 0x54
[[0x00000054]> pf.elf_phdr
    type : 0x00000054 = type (enum elf_p_type) = 0x1 ; PT_LOAD
  offset : 0x00000058 = 0x00000448
   vaddr : 0x0000005c = 0x0805f448
   paddr : 0x00000060 = 0x0805f448
  filesz : 0x00000064 = 0x00000000
   memsz : 0x00000068 = 0x00000000
   flags : 0x0000006c = flags (enum elf_p_flags) = 0x6 ; PF_Read_Write
   align : 0x00000070 = 0x00001000
[[0x00000054]>
 [0x00000054]>
```

Load sample with -nn; Goto 1st or 2nd header; Run: pf.elf_phdr

How about the program header entries? (elfdump)

```
entry: 0
    p_type: PT_LOAD
    p_offset: 0
    p_vaddr: 0xc01000
    p_paddr: 0xc01000
    p_filesz: 34856
    p_memsz: 34856
    p_flags: PF_X|PF_R
    p_align: 4096

entry: 1
    p_type: PT_LOAD
    p_offset: 1096
    p_vaddr: 0x805f448
    p_paddr: 0x805f448
    p_filesz: 0
    p_memsz: 0
    p_flags: PF_W|PF_R
    p_align: 4096
```

Use: BSD command elfdump to dump the ELF header

To do manual analysis of this ELF header in radare2 interface correctly

Run: / ELF; Seek for ELF magic; s <address> ; Run: pxa

...and so many unlimited variations of these ELF too

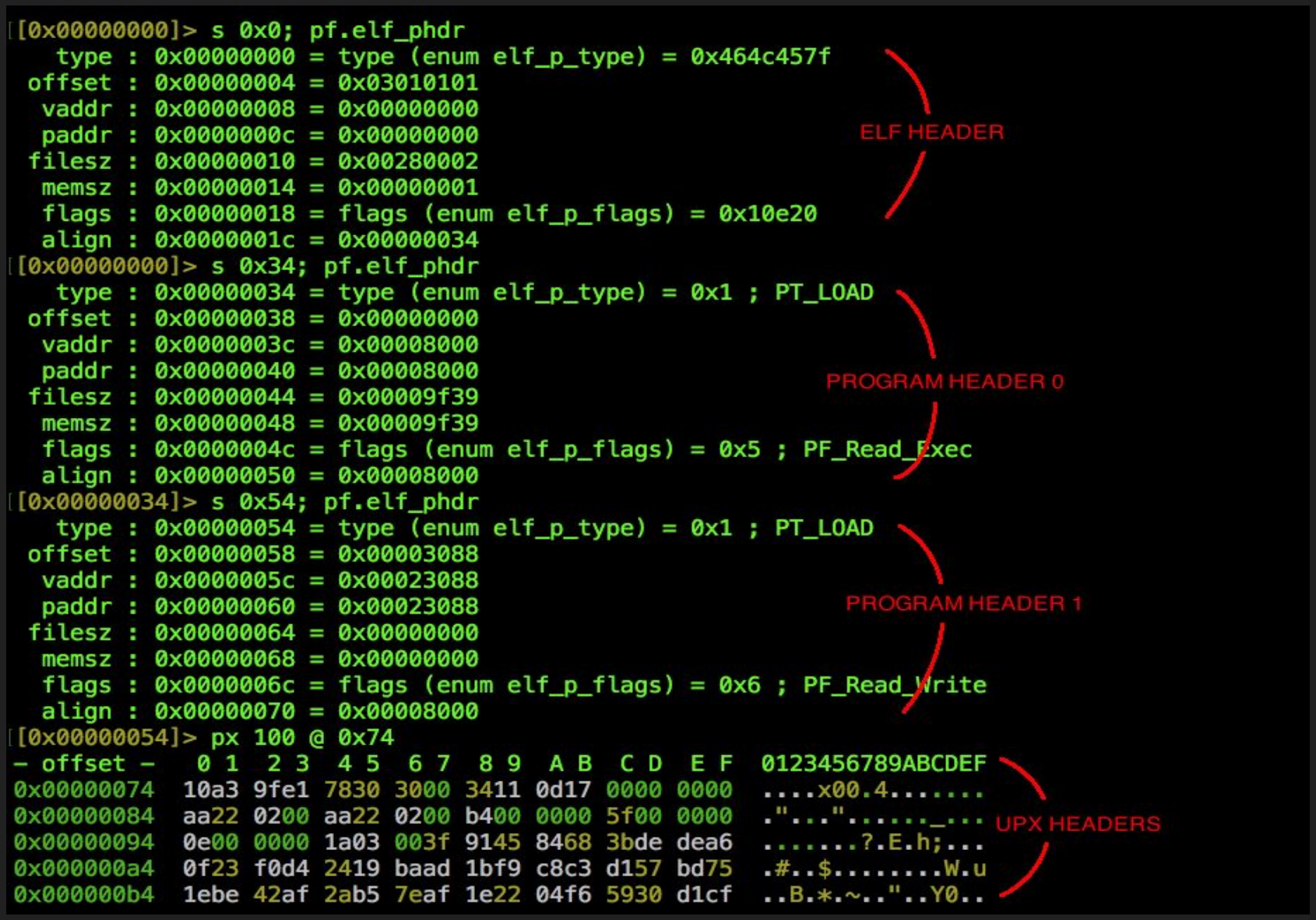
```
0x00008001 hit0_0 .ELF(.
[0x00010e20]> s 0x00008000
[0x00008000]> x
- offset -   0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0123456789ABCDEF
0x00008000  7f45 4c46 0101 0103 0000 0000 0000 0000  .ELF............
0x00008010  0200 2800 0100 0000 200e 0100 3400 0000  ..(..... ...4...
0x00008020  0000 0000 0200 0004 3400 2000 0200 2800  ........4. ...(.
0x00008030  0000 0000 0100 0000 0000 0000 0080 0000  ................
0x00008040  0080 0000 399f 0000 399f 0000 0500 0000  ....9...9.......
0x00008050  0080 0000 0100 0000 8830 0000 8830 0200  .........0...0..
0x00008060  8830 0200 0000 0000 0000 0000 0600 0000  .0..............
0x00008070  0080 0000 10a3 9fe1 7830 3000 3411 0d17  ........x00.4...
0x00008080  0000 0000 aa22 0200 aa22 0200 b400 0000  ....."..."......
0x00008090  5f00 0000 0e00 0000 1a03 003f 9145 8468  _..........?.E.h
0x000080a0  3bde dea6 0f23 f0d4 2419 baad 1bf9 c8c3  ;....#..$.......
0x000080b0  d157 bd75 1ebe 42af 2ab5 7eaf 1e22 04f6  .W.u..B.*.~.."..
0x000080c0  5930 d1cf 45f3 0071 9918 a740 d183 d47b  Y0..E..q...@...{
0x000080d0  6314 08f3 7b75 c7cd d3b6 7467 57b5 989c  c...{u....tgW...
0x000080e0  eb94 d5a6 20d5 fe17 9698 1fce 0661 1aff  .... .........a..
0x000080f0  e65e 7de9 2ed5 e0e8 7e01 0024 8b00 000e  .^}.....~..$....
[0x00008000]>
```

Anyone can tell what's wrong with this ELF?

Practise to read ELF(upx) headers w/ radare2 (default)

```
 -- Jingle sploits, jingle sploits, ropchain all the way.
[0x00010e20]> / ELF
Searching 3 bytes in [0x8000-0x11f39]
hits: 1
0x00008001 hit0_0 .ELF(.
[0x00010e20]> 0x00008000
[0x00008000]> pxa
- offset -   0 1  2 3  4 5  6 7  8 9  A B  C D  E F
            /s/hit0_0ehdr
0x00008000  7f45 4c46 0101 0103 0000 0000 0000 0000  .ELF............
0x00008010  0200 2800 0100 0000 200e 0100 3400 0000  ..(..... ...4...
0x00008020  0000 0000 0200 0004 3400 2000 0200 2800  ........4. ...(.
                     /segment_end.ehdr
0x00008030  0000 0000 0100 0000 0000 0000 0080 0000  ................
0x00008040  0080 0000 399f 0000 399f 0000 0500 0000  ....9...9.......
0x00008050  0080 0000 0100 0000 8830 0000 8830 0200  .........0...0..
0x00008060  8830 0200 0000 0000 0000 0000 0600 0000  .0..............
0x00008070  0080 0000 10a3 9fe1 7830 3000 3411 0d17  ........x00.4...
0x00008080  0000 0000 aa22 0200 aa22 0200 b400 0000  ....."..."......
0x00008090  5f00 0000 0e00 0000 1a03 003f 9145 8468  _..........?.E.h
0x000080a0  3bde dea6 0f23 f0d4 2419 baad 1bf9 c8c3  ;....#..$.......
0x000080b0  d157 bd75 1ebe 42af 2ab5 7eaf 1e22 04f6  .W.u..B.*.~.."..
```

Practise to read ELF(upx) headers with radare2 -n



...checking whether the values are correct (elfdump)

```
e_ident: ELFCLASS32 ELFDATA2LSB ELFOSABI_LINUX
e_type: ET_EXEC
e_machine: EM_ARM
e_version: EV_CURRENT
e_entry: 0x10e20
e_phoff: 52
e_shoff: 0
e_flags: 67108866
e_ehsize: 52
e_phentsize: 32
e_phnum: 2
e_shentsize: 40
e_shnum: 0
e_shstrndx: 0

entry: 0
    p_type: PT_LOAD
    p_offset: 0
    p_vaddr: 0x8000
    p_paddr: 0x8000
    p_filesz: 40761
    p_memsz: 40761
    p_flags: PF_X|PF_R
    p_align: 32768

entry: 1
    p_type: PT_LOAD
    p_offset: 12424
    p_vaddr: 0x23088
    p_paddr: 0x23088
    p_filesz: 0
    p_memsz: 0
    p_flags: PF_W|PF_R
    p_align: 32768
```

Use: BSD command elfdump to dump the ELF header

Back to the question:

" So, what is wrong with those files? "

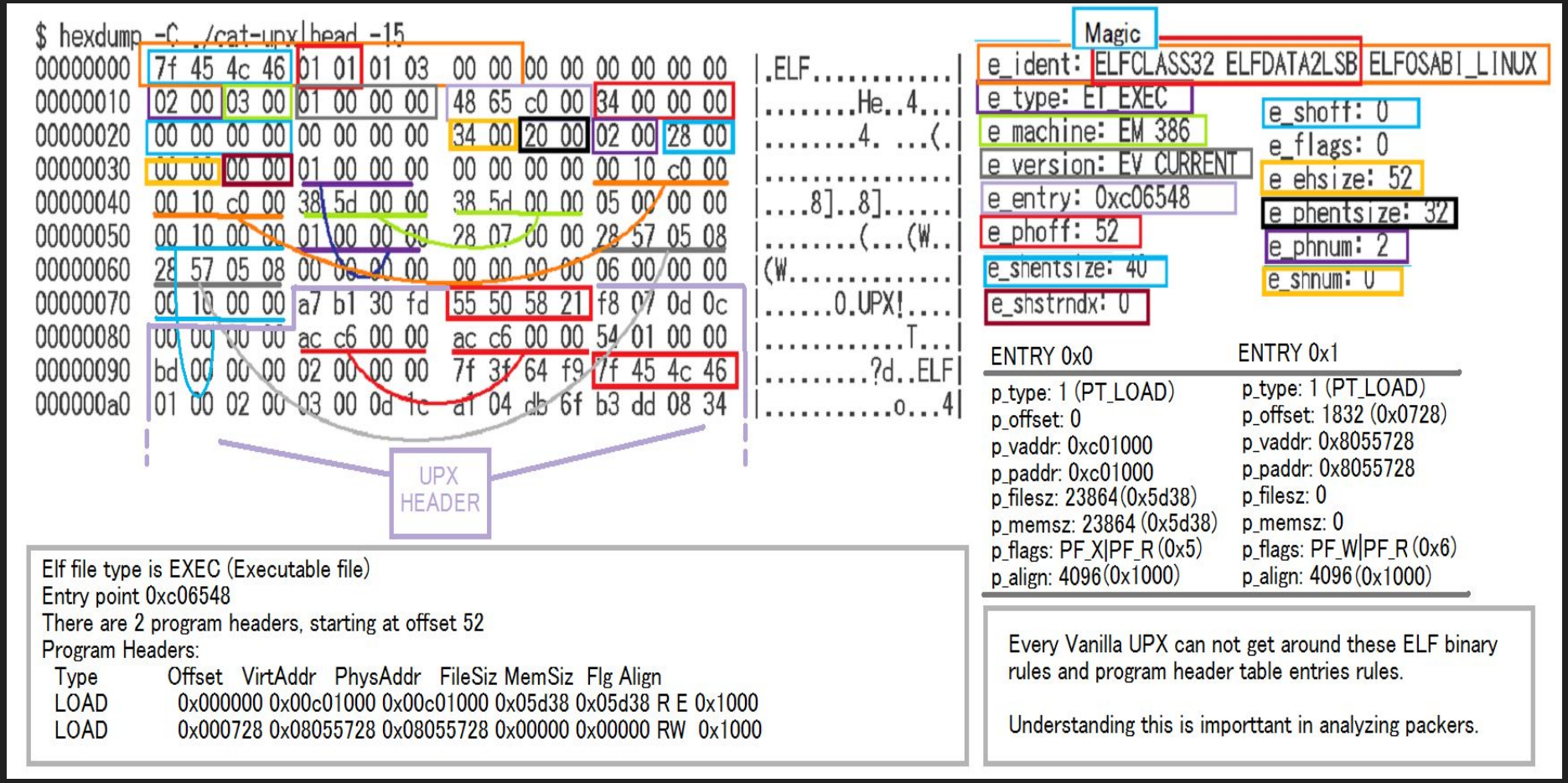
Motto of the day

Remember our motto of the day:

“ FIRE UP YOUR R2!

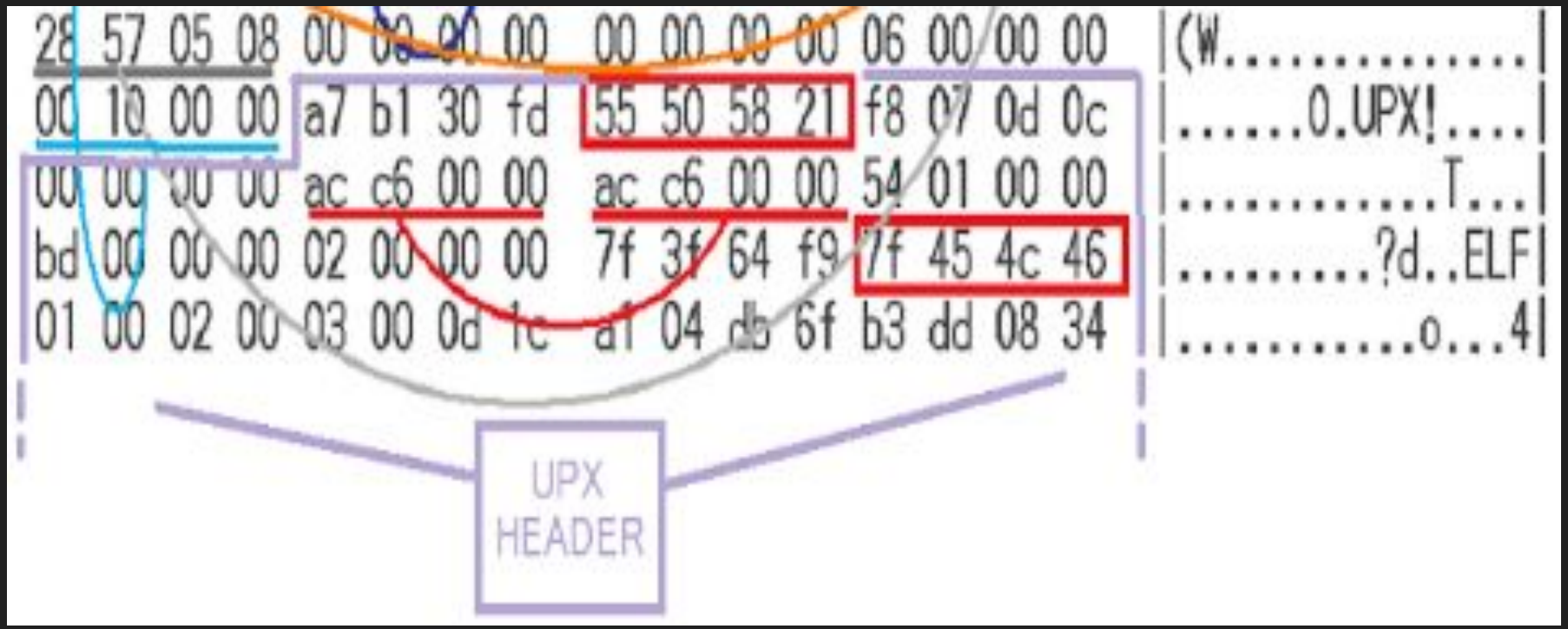
+ ..HAVE FAITH! DON’T GIVE UP ! "

Remember this pic when dealing with Vanilla UPX



..above image is for ELF UPX x86_32, use this concept to recognize the headers for the other architecture too, then write your own image.

Especially this part, the one that has to be correctly written for decompressing an ELF UPX



..and (hopefully) all will be okay :)

Fixing the headers i.e.:

```
0x00c01070  0010 0000 d9ba 2ad2 5550 5821 1008 0d0c  ......*.UPX!....
0x00c01080  0000 0000 8c08 0100 8c08 0100 b400 0000  ................
0x00c01090  7200 0000 0200 0000 7f3f 64f9 7f45 4c46  r.........?d..ELF
0x00c010a0  0100 0200 0300 0d94 8104 fe6f b3dd 0834  ...........o...4
0x00c010b0  0734 061b 0b20 0004 0028 000f 000e 005f  .4... ...(....._
0x00c010c0  fbb5 c92d 8023 0313 ea0a 0305 7dee ce7e  ...-.#......}..~
```

```
0x00c01090  7200 0000 0200 0000 7f3f 64f9 7f45 4c46  r.........?d..ELF
0x00c010a0  0100 0200 0300 0d94 8104 fe6f b3dd 0834  ...........o...4
0x00c010b0  0734 061b 0b20 0004 0028 000f 000e 005f  .4... ...(....._
[0x00c01000]> q
[          /malware/temp/028/          /005]$ upx -d pty2
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2013
UPX 3.91        Markus Oberhumer, Laszlo Molnar & John Reiser   Sep 30th 2013

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
     69241 <-     34916   50.43%  netbsd/elf386  pty2

Unpacked 1 file.
```

What if things are still gone wrong?

```
[0x00000000]> # in here..
[0x00000000]> px @0x59!0x1
- offset -   0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0123456789ABCDEF
0x00000059  0d                                      .
[0x00000000]> # If you see this blob....
[0x00000000]> px @0x58!0x2
- offset -   0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0123456789ABCDEF
0x00000058  680d                                    h.
[0x00000000]> # it is not suppose to be the same as:
[0x00000000]> px @0x5C!0x2
- offset -   0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0123456789ABCDEF
0x0000005c  680d                                    h.
[0x00000000]> #This bit was faked:
[0x00000000]> px @0x59!0x1
- offset -   0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0123456789ABCDEF
0x00000059  0d                                      .
[0x00000000]> # so putting the correct bit together will restore the ELF
[0x00000000]> # however if one doesn't know the orig ELF form it is impo-
[0x00000000]> #ssible.. what to do?
[0x00000000]> #
```

Note from pancake: To be more readable please use @0x5c!1

..and (hopefully) all will be okay :)

Brute it!

```
[0x00cfce38]>
[0x00cfce38]>
[0x00cfce38]> px @0x59!0x1
- offset -   0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0123456789ABCDEF
0x00000059  0d                                      .
[0x00cfce38]> # patch it..
[0x00cfce38]> oo+; s 0x59; px 1; wx 00; px 1
File stdin-dep reopened in read-write mode
Warning: Cannot initialize section headers
Warning: Cannot initialize strings table
Warning: Cannot initialize dynamic strings
- offset -   0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0123456789ABCDEF
0x00000059  0d                                      .
- offset -   0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0123456789ABCDEF
0x00000059  00                                      .
```

Video: https://www.youtube.com/watch?v=72JpU95aDLg

.and (hopefully) all will be okay :)

Vanilla UPX - Fixing the hacked source code

FACTS:

- ELF packed with UPX and the headers looks okay
- Can NOT be decompress even all are in correct places.

SOLUTION:

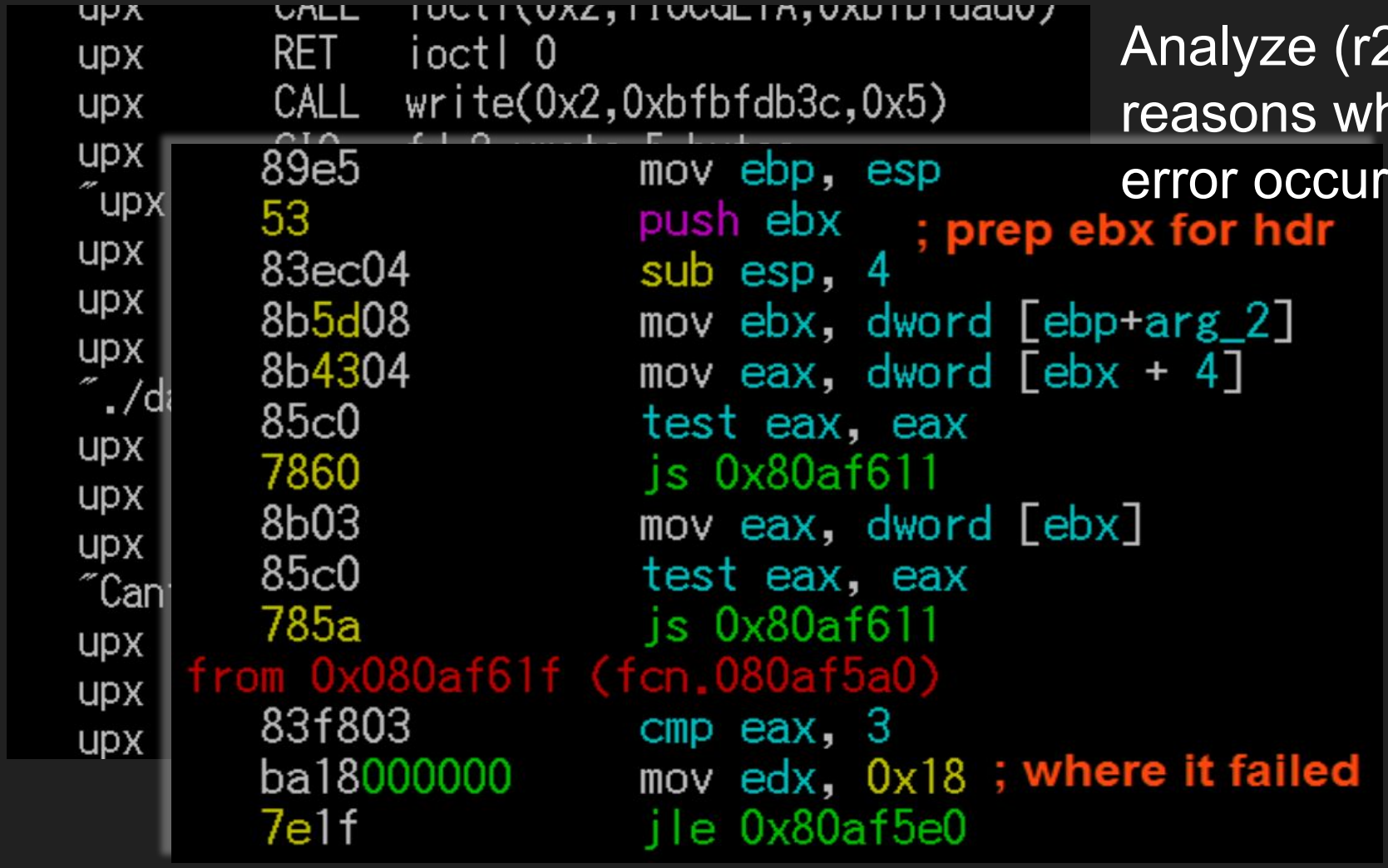
- Debug until point where the decompressor error come up
- Find how the error behavior is and seek which variable is causing it
- Go to the official UPX source code to see if it is correctly coded that way
- Diff it, see how it is patched and adjust the source code, recompile the UPX and try to decompress with the new fixed source code

Vanilla UPX - Fixing the hacked source code

```
upx      CALL  ioctl(0x2,PROCGETA,0xbfbfdad0)
upx      RET   ioctl 0
upx      CALL  write(0x2,0xbfbfdb3c,0x5)
upx      GIO   fd 2 wrote 5 bytes
"upx: "
upx      RET   write 5
upx      CALL  write(0x2,0xbfbfdb3c,0x15)
upx      GIO   fd 2 wrote 21 bytes
"./daemon.armv4l.mod: "
upx      RET   write 21/0x15
upx      CALL  write(0x2,0xbfbfdf6c,0x27)
upx      GIO   fd 2 wrote 39 bytes
"CantUnpackException: header corrupted 2"
upx      RET   write 39/0x27
upx      CALL  write(0x2,0x81b8640,0x1)
upx      GIO   fd 2 wrote 1 byte
```

Output of the ktrace read in kdump (on BSD)

Vanilla UPX - Fixing the hacked source code



Analyze (r2) reasons why the error occured ...

Vanilla UPX - Fixing the hacked source code

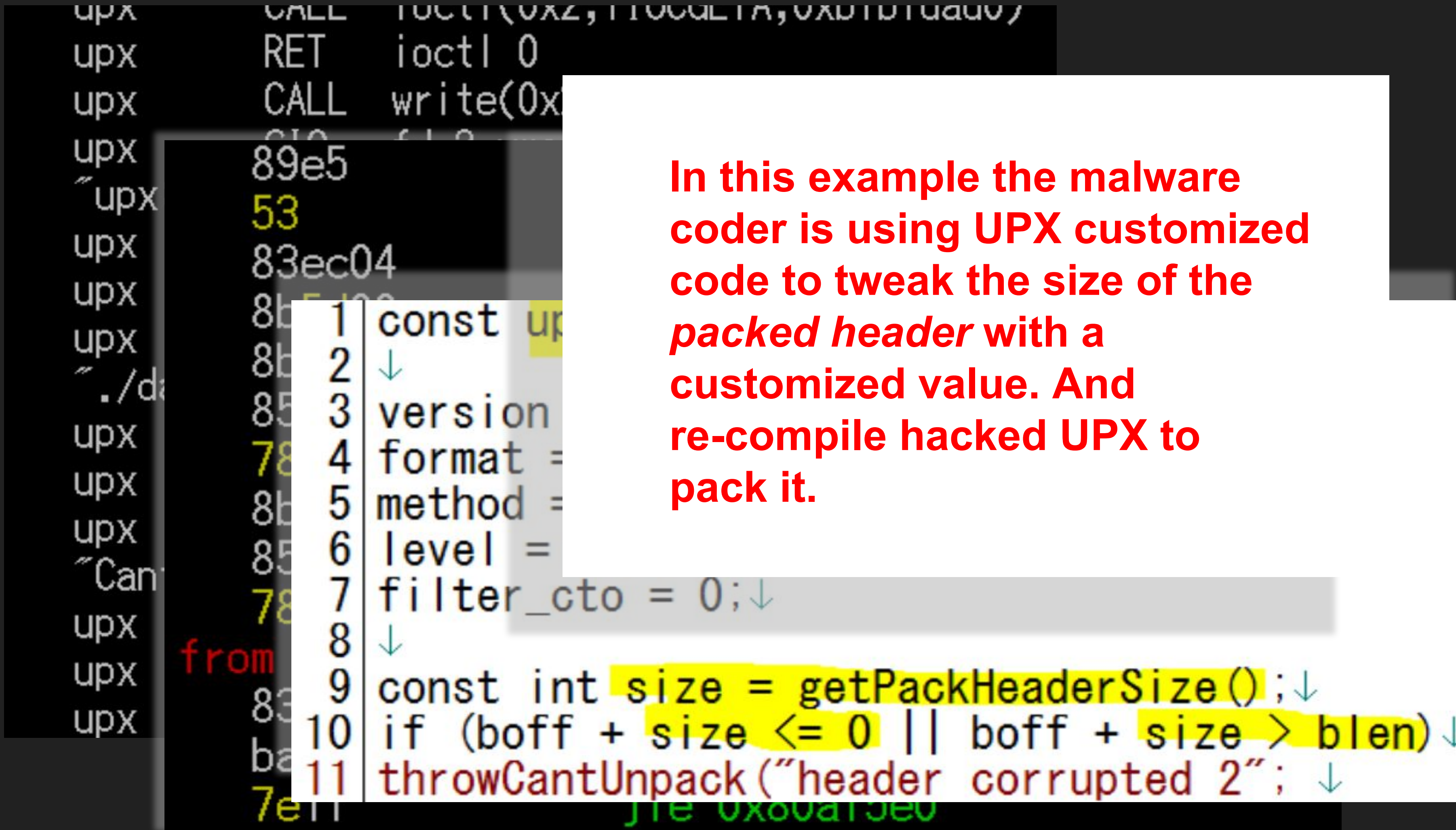
```
upx     RET   ioctl 0
upx     CALL  write(0x2,0xbfbfdb3c,0x5)
upx     89e5          mov ebp, esp
upx     53            push ebx    ; prep ebx for hdr
upx     83ec04        sub esp, 4
```

```
 1 const upx_bytep p = buf + boff;↓
 2 ↓
 3 version = p[4];↓
 4 format = p[5];↓
 5 method = p[6];↓
 6 level = p[7];↓
 7 filter_cto = 0;↓
 8 ↓
 9 const int size = getPackHeaderSize();↓
10 if (boff + size <= 0 || boff + size > blen)↓
11 throwCantUnpack("header corrupted 2"; ↓
```

Check which UPX source was tweaked..

More explanation in: https://imgur.com/a/3mgZm/

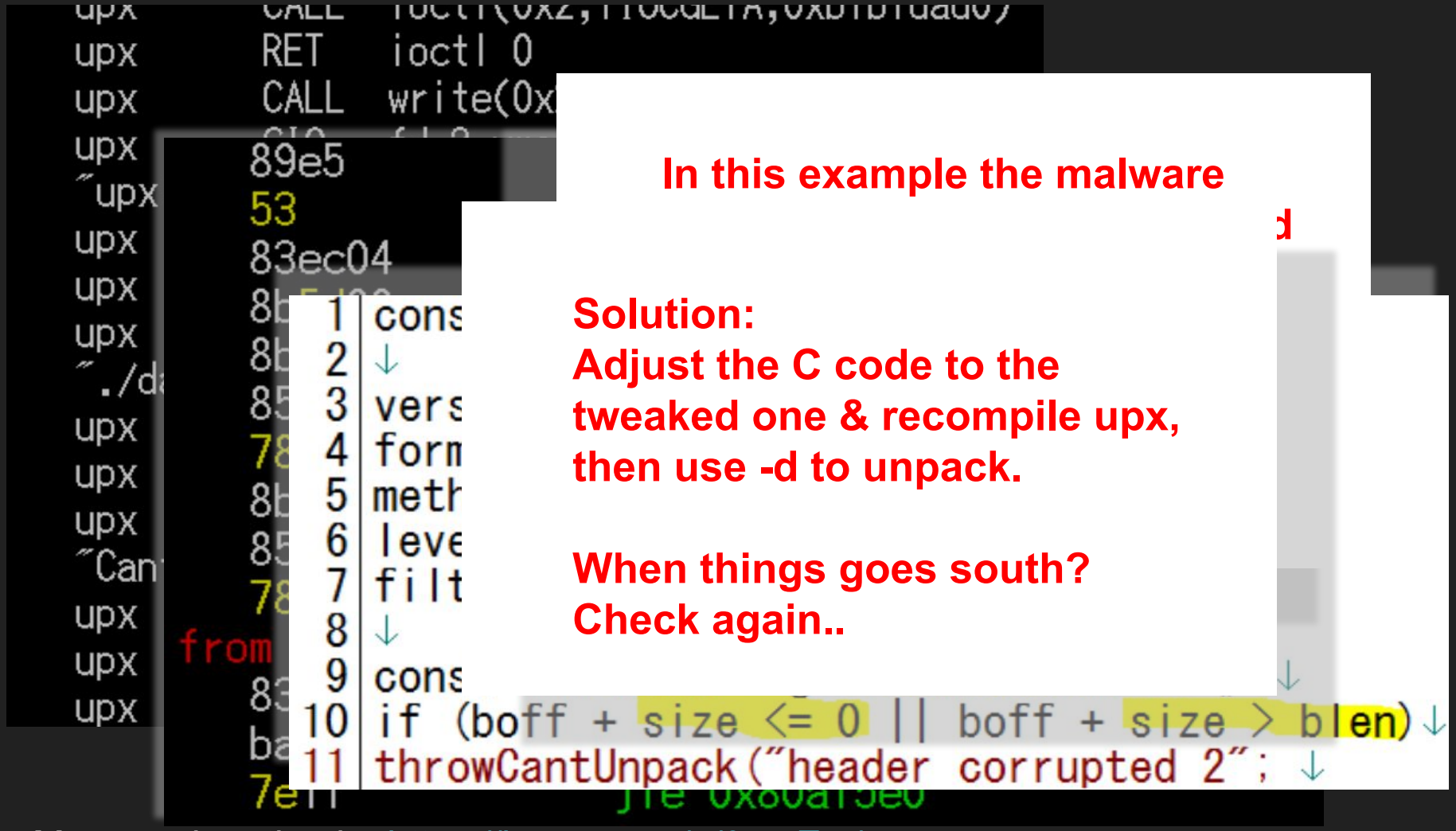
Vanilla UPX - Fixing the hacked source code



In this example the malware coder is using UPX customized code to tweak the size of the *packed header* with a customized value. And re-compile hacked UPX to pack it.

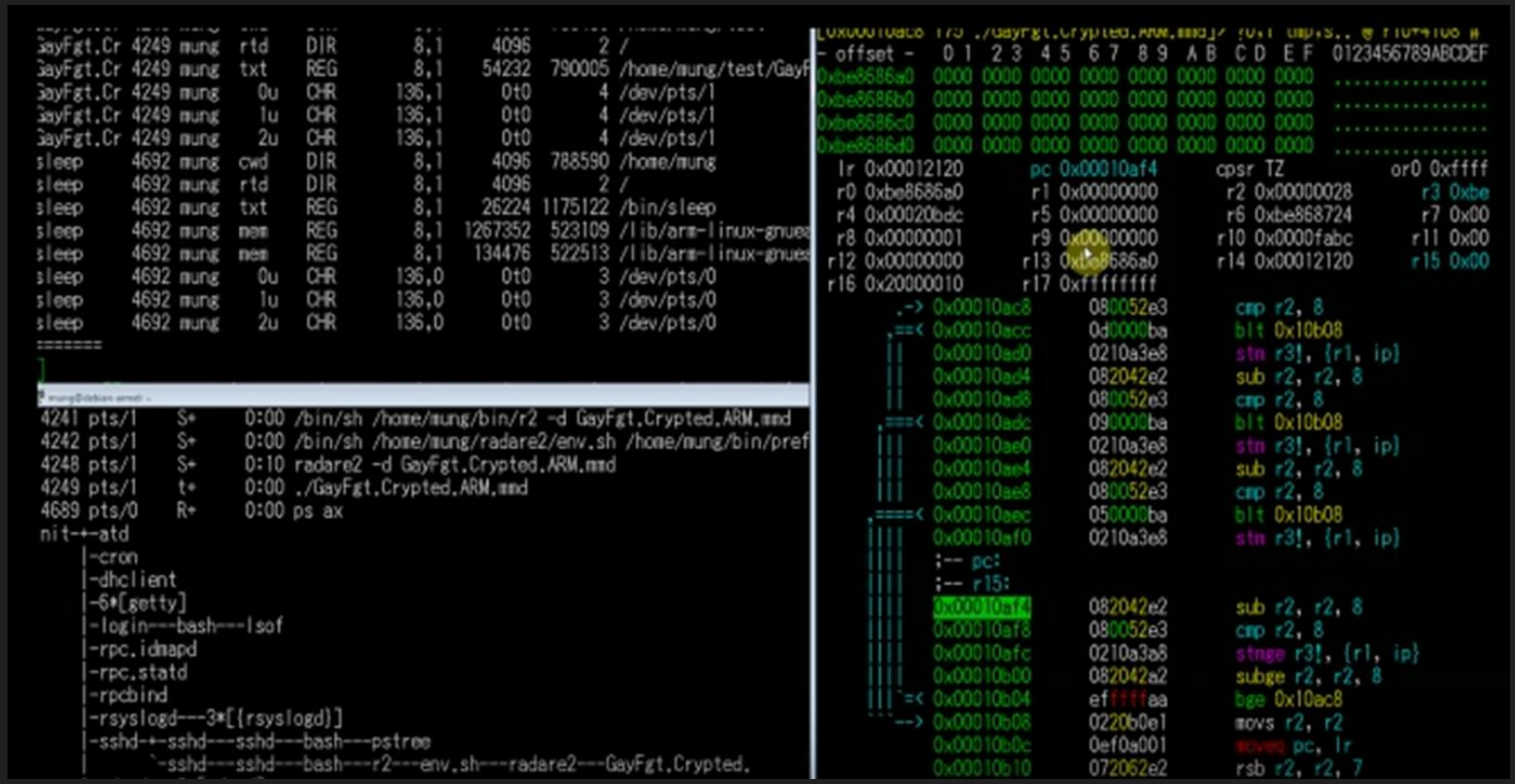
More explanation in: https://imgur.com/a/3mgZm/

Vanilla UPX - Fixing the hacked source code



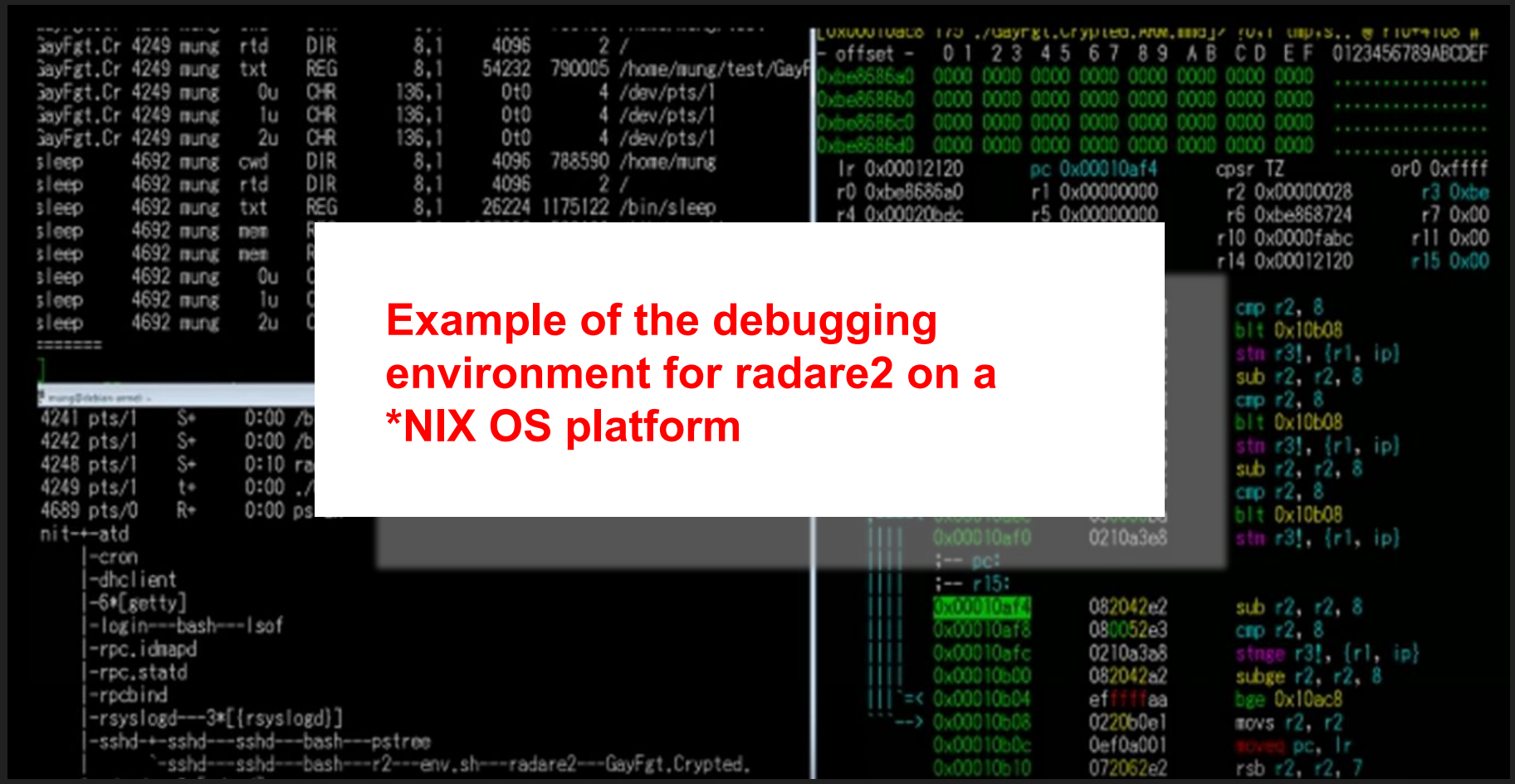
More explanation in: https://imgur.com/a/3mgZm/

Vanilla UPX - On memory un-packing with r2



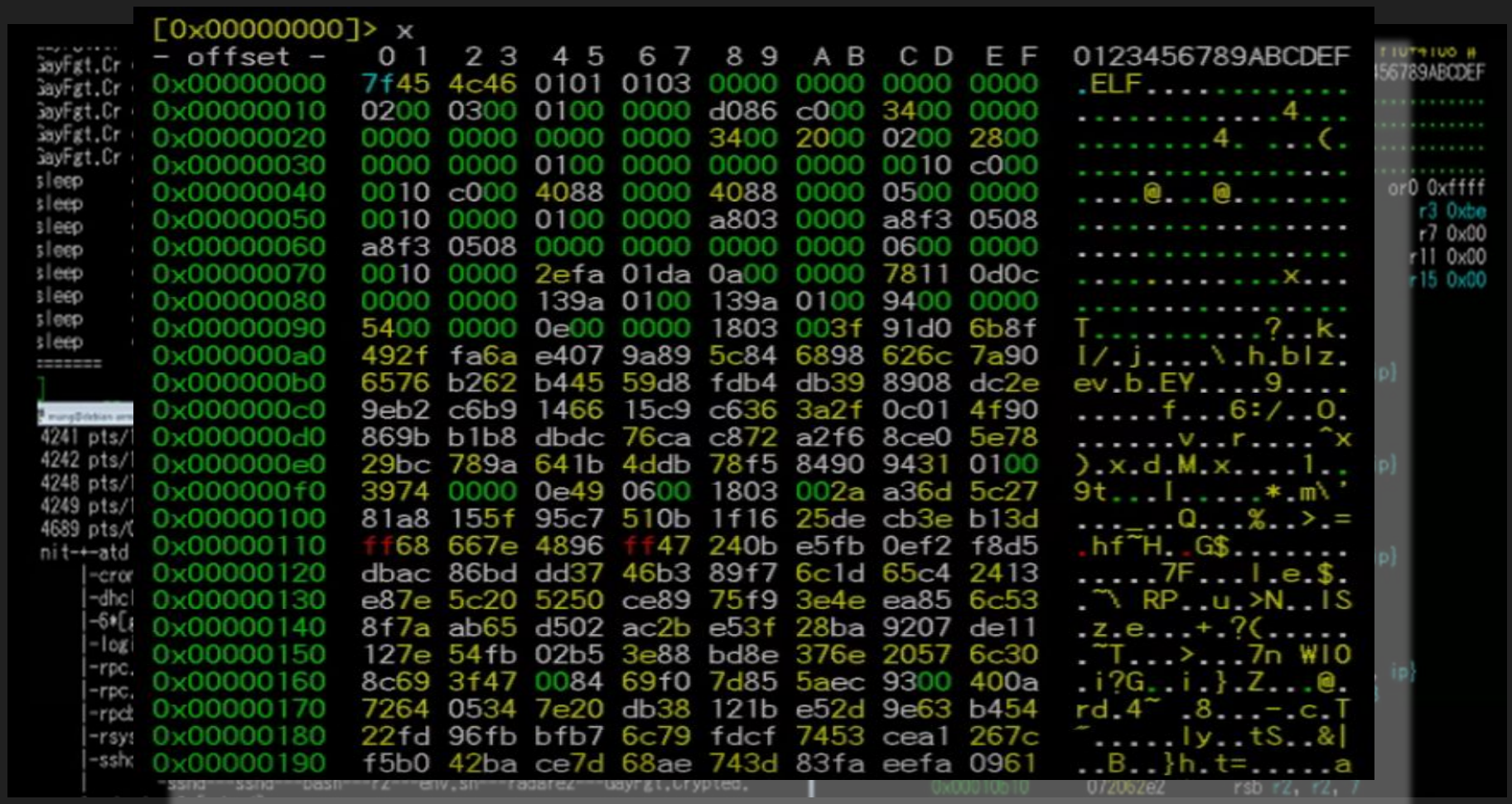
Video: https://www.youtube.com/watch?v=ad6kvL834Wk

Vanilla UPX - On memory un-packing with r2



Video: https://www.youtube.com/watch?v=ad6kvL834Wk

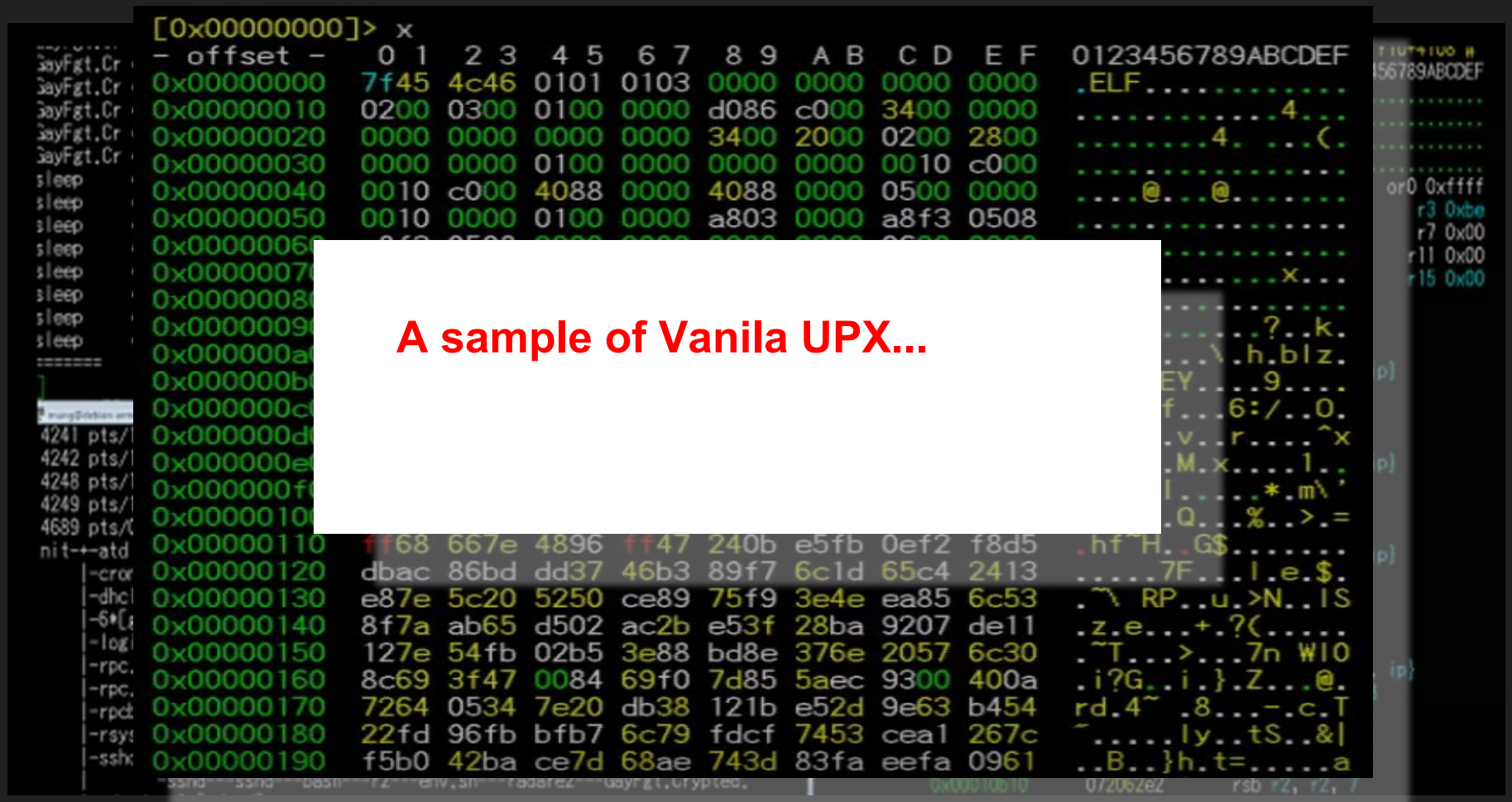
Vanilla UPX - On memory un-packing with r2



Video: https://www.youtube.com/watch?v=ad6kvL834Wk

https://www.youtube.com/watch?v=E8gN_XUTCxM

Vanilla UPX - On memory un-packing with r2



Video: https://www.youtube.com/watch?v=ad6kvL834Wk

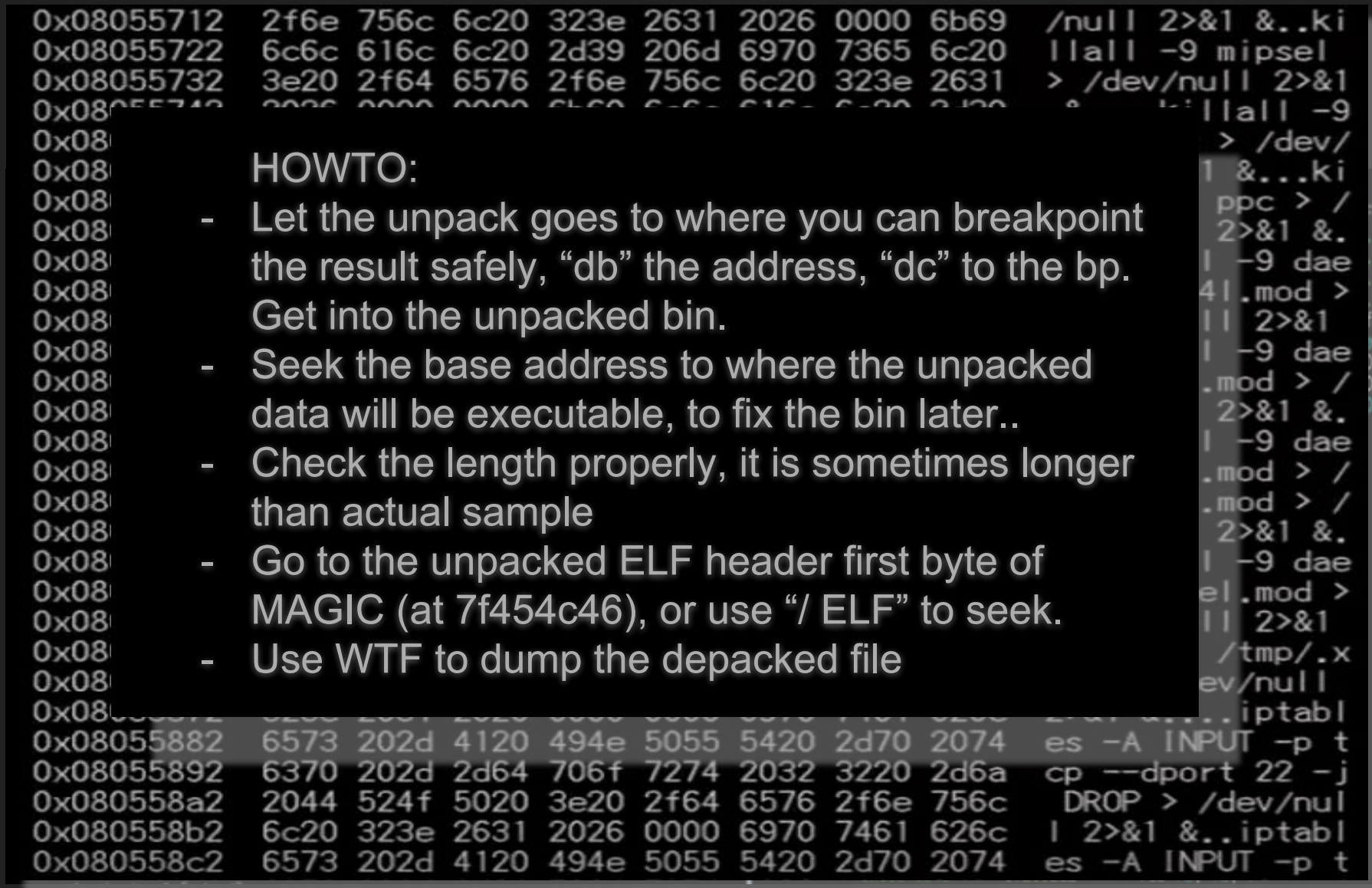
https://www.youtube.com/watch?v=E8gN_XUTCxM

Video: https://www.youtube.com/watch?v=ad6kvL834Wk

https://www.youtube.com/watch?v=E8gN_XUTCxM

Video: https://www.youtube.com/watch?v=ad6kvL834Wk

https://www.youtube.com/watch?v=E8gN_XUTCxM



HOWTO:

- Let the unpack goes to where you can breakpoint the result safely, “db” the address, “dc” to the bp. Get into the unpacked bin.
- Seek the base address to where the unpacked data will be executable, to fix the bin later..
- Check the length properly, it is sometimes longer than actual sample
- Go to the unpacked ELF header first byte of MAGIC (at 7f454c46), or use “/ ELF” to seek.
- Use WTF to dump the depacked file

Video: https://www.youtube.com/watch?v=ad6kvL834Wk
https://www.youtube.com/watch?v=E8gN_XUTCxM
(to be added…)

Several Vanilla UPX samples to practise

Binaries info:

1. 8046e04564dc0b8d0e5a37358d6a8dfe
   ELF 32-bit LSB executable, ARM, version 1 (ARM), statically linked, stripped

2. 7980ffb3ad788b73397ce84b1aadf99b
   ELF 32-bit LSB executable, MIPS, MIPS-I version 1 (SYSV), statically linked, stripped

Motto of the day

..SO EVERYTHING IS A "BYTE”
OKAY NOW! :)

This is how you should feel:

Yay, we can beat some Vanillas..

Other ELF packers I researched so far..

Summarizing:

1. BurnEye (good for practise, has crypted password protection, basic of packing in ELF, Open source/Leaked Source?)
2. ElfEncrypter/ElfJunk (virtualize then obfuscate..and obfuscate, has password protection, Open source)
3. Shiva (The first anti reversing packer I tested, a bit old but one of best I guess, not seeing this anymore ITW, except older samples, hard to dissect with only static analysis)
4. VMProtect (Use Virtual Machine, Obfuscation code, Highly used of Junk Assembly, With several anti debugging, please read official web site for more, currently I am on watch this packer)

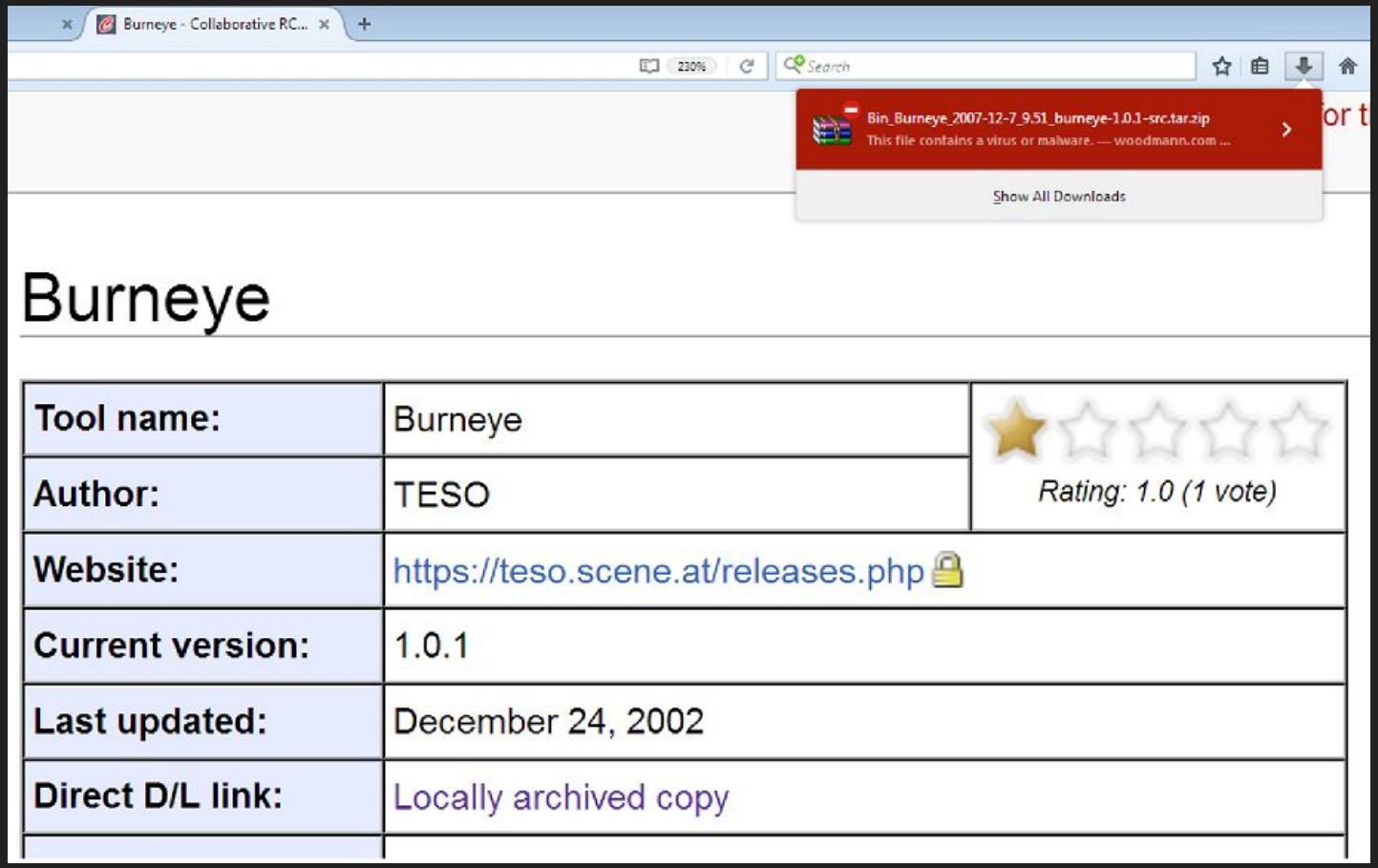
Other ELF packers I researched so far..

TABLE OF PACKER DATA	Origin	Features (Crypt etc)	Dificulty score	Dissection info	Arch	Reference
BurnEye	2002	Password protected (TESO Elf Encrypt Engine)	low	burndump.c - burneye unwrapper by [ByteRage]	x86	woodmann.com/collaborative/tools/index.php/Burneye
ElfCrypter/ ElfJunk	2004-2005	Encryption with crypt-7lib Obfuscation	low	Script & manual	x86	elf-encrypter.sourceforge.net
Shiva	Shiva-0.96 by Clowes & Mehta	Packed ELF crypted w/ AES/ 128 Passw=SHA1 hash	Hard, several anti reversing & adapt BurnEye	Manual	x86	secureReality.com.au/archives/shiva-0.95.tar.gz
VMProtect	last test: 2017/ Vmpsoft	VM protect Junk ASM Anti reverse	Very hard	Manual	x86_32 x86_64	VMProtect Demo for Linux 3.1.2.946

Other ELF packers I researched so far..

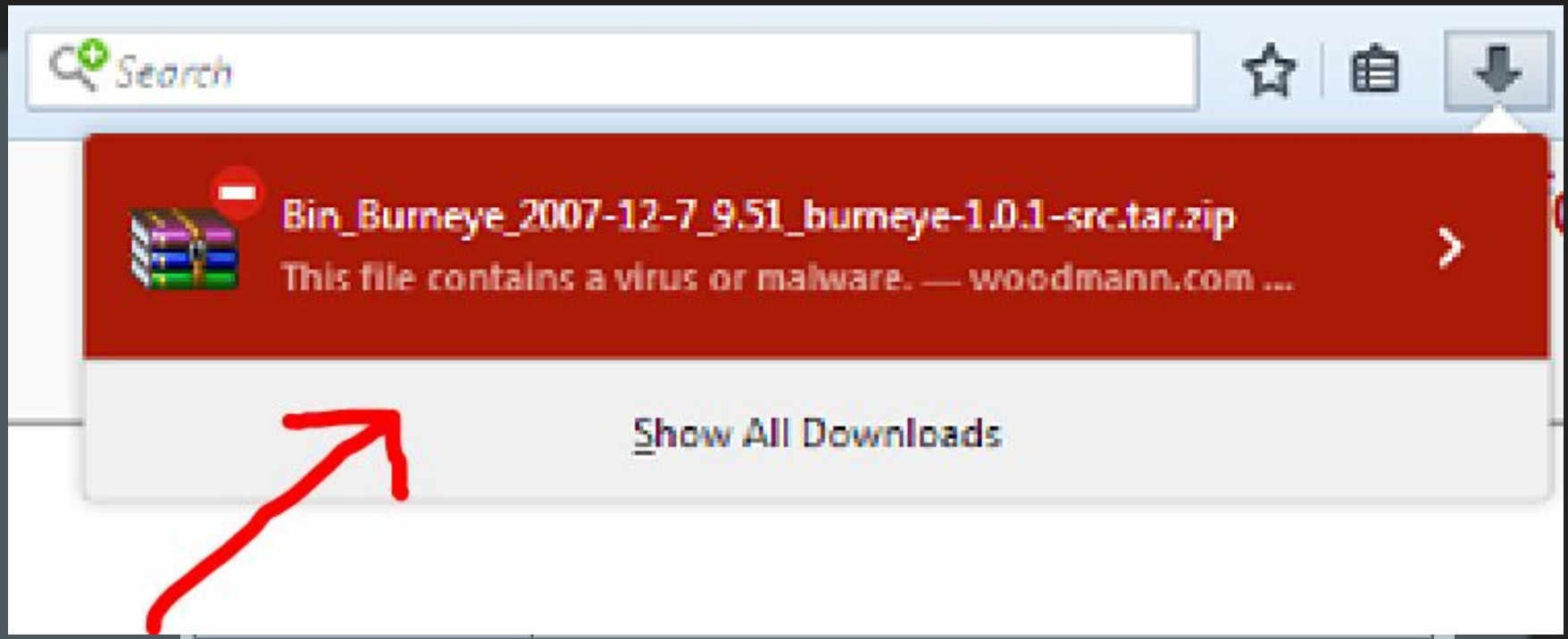
Be careful in downloading known ancient packer's source code, your network security sensors may raise alarm! Some sigs just can't differ..

PoC:



Other ELF packers I researched so far..

Be careful in downloading known packer's source code, your network security sensors may raise alarm!



Last updated:	December 24, 2002
Direct D/L link:	Locally archived copy

Other ELF packers I researched so far..

ELF-Encrypter home page

The ELF-Encrypter program suite is a collection of programs to encrypt ELF binaries using various algorithms, including the ones provided by *GPG*.
One can choose a lot of methods to obtain the encryption key, such as hashing a list of PCI peripheals, MAC addresses of ethernet cards, file inode numbers, passphrases and passwords.

Here is the complete list:

- 8-bit fixed mask (used only for testing)
- 32-bit user selectable mask (used only for testing)
- Password
- Cast5 (using GPG)
- GPG - Encrypt (using GPG)
- GPG - Sign (using GPG)
- Ethernet MAC
- Ethernet MAC and file inode
- File inode
- PCI binding

Executable files can be encrypted using the *crypt-7lib* program, then decrypted at runtime by a shared library (*libdecrypt*), directly linked to the binary or listed in LD_PRELOAD.
Users are strongly encouraged to use *gpg-agent*; however a special support for *quintuple-agent* is available at compilation time.

The suite also contains programs to manipulate and inject plain or encrypted code into ELF binaries.

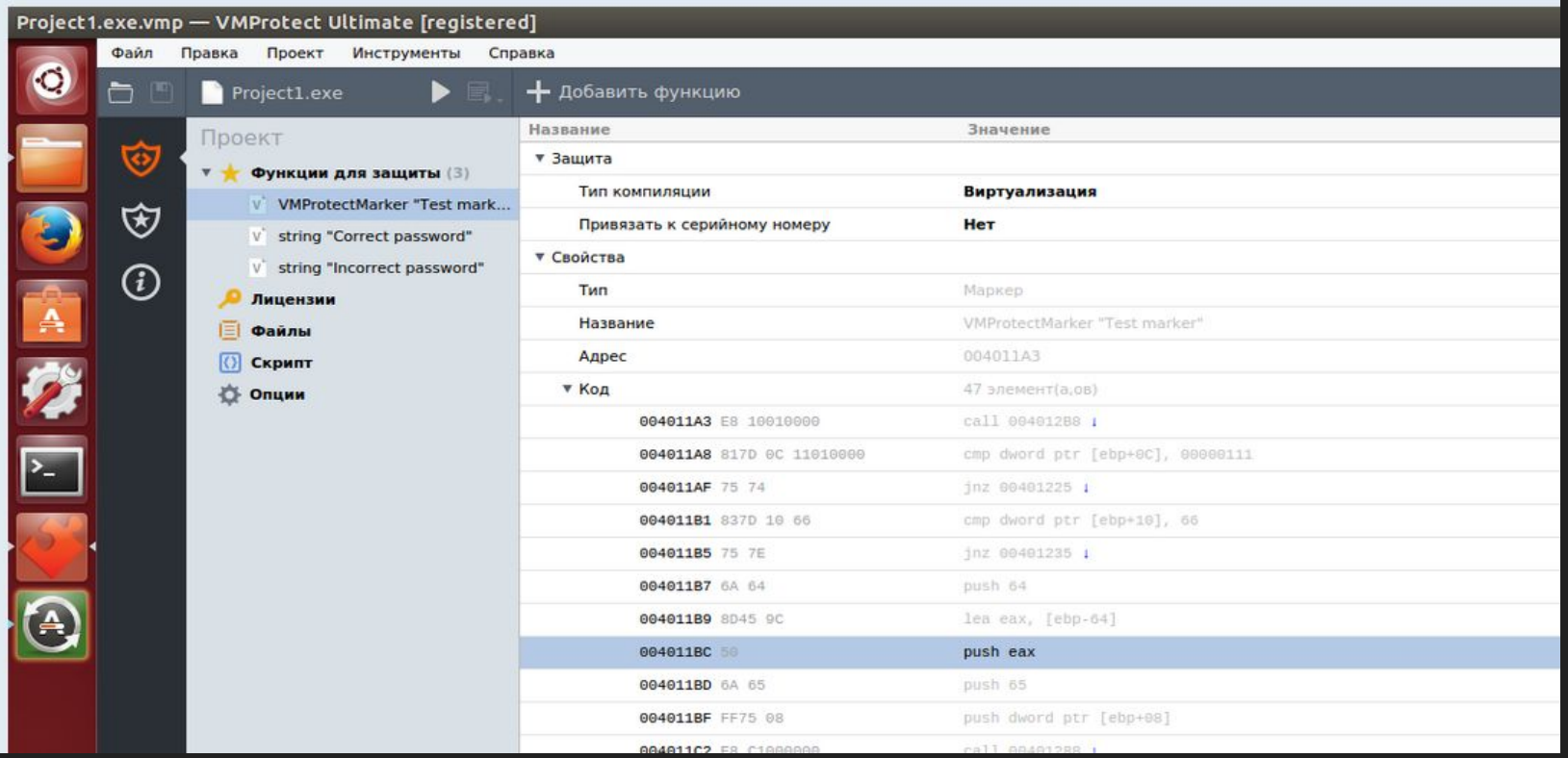
This project is written in C and assembly for the x86 platform.

Other ELF packers I researched so far..

VMProtect for Linux

by **Admin** » Tue Apr 18, 2017 6:28 pm

VMProtect Demo for Linux 3.1.2.946:
http://vmpsoft.com/files/VMProtectDemo_x64.tar.gz



Part two - some soup

“ Interesting ELF packers today
(A good take-away for you) “

Interesting “cool” ELF packers exist today

1. arisada/midgetpack (feature: Curve25519 - challenge-response)
   https://github.com/arisada/midgetpack
2. glen-mac/ELF-Packer (feature: Polymorphic python packer ELF)
   https://github.com/glen-mac/ELF-Packer
3. sebastiencs/Packer_ELF (simplistic, for learning or base of dev)
   https://github.com/sebastiencs/Packer_ELF
4. ps2dev/ps2-packer (designed for PS2, and modular design)
   https://github.com/ps2dev/ps2-packer
5. timhsutw/elfuck (using NRV2E decompressor, string burner polymorphic & adapting BurnEye locking feature)
   https://github.com/timhsutw/elfuck

...see its URL above for the further details, they are so self explanatory, I made review on each of them as per written above after my tests.

Motto of the day

ARE YOU STILL OKAY?

Now you now some cool packer's too!
..but some of you may feel like:

"Okay! I know more packers than you!
Enough talking & gimme THAT main course!! "

Part three - the main course

“ The story of :
a Non-Unpackable ELF Packer.. “

What I want to share today (about this N.U.P)

1. I don't know the source for this packer yet, I will call it with a dumb-pseudo-name: Non-Unpackable Packer (N.U.P) until we know exactly what the real name is...
2. Spotted in the public threat sector (+ "maybe" some targeted sectors)
3. It aims 3.1.) fileless forensics and 3.2) obviously hiding its packed binary to make sure even static analysis can not break it..

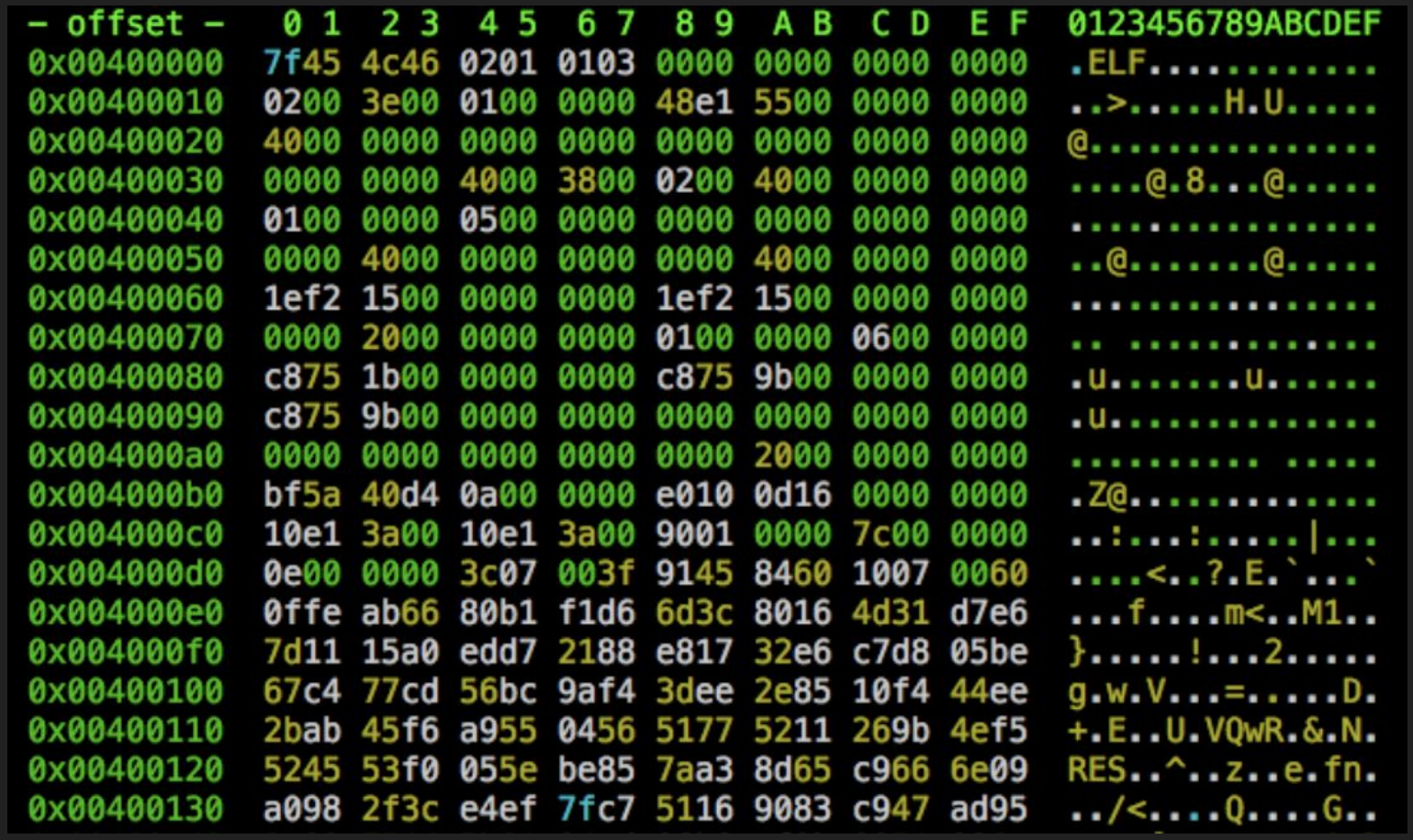
Why you called it “non-unpackable” ?

1. I found this hard to be dissected statically
2. It was designed to be anti-emulator (in our case is ESIL) with cascade chains of obfuscation
3. It’s actually a custom made from one of previous packers mentioned..

Here we go . .

Non Unpackable Packer

The sample is an ITW threat, and goes with this ELF headers:



```
- offset -   0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0123456789ABCDEF
0x00400000  7f45 4c46 0201 0103 0000 0000 0000 0000  .ELF............
0x00400010  0200 3e00 0100 0000 48e1 5500 0000 0000  ..>.....H.U.....
0x00400020  4000 0000 0000 0000 0000 0000 0000 0000  @...............
0x00400030  0000 0000 4000 3800 0200 4000 0000 0000  ....@.8...@.....
0x00400040  0100 0000 0500 0000 0000 0000 0000 0000  ................
0x00400050  0000 4000 0000 0000 0000 4000 0000 0000  ..@.......@.....
0x00400060  1ef2 1500 0000 0000 1ef2 1500 0000 0000  ................
0x00400070  0000 2000 0000 0000 0100 0000 0600 0000  .. ............
0x00400080  c875 1b00 0000 0000 c875 9b00 0000 0000  .u.......u......
0x00400090  c875 9b00 0000 0000 0000 0000 0000 0000  .u..............
0x004000a0  0000 0000 0000 0000 0000 2000 0000 0000  .......... .....
0x004000b0  bf5a 40d4 0a00 0000 e010 0d16 0000 0000  .Z@.............
0x004000c0  10e1 3a00 10e1 3a00 9001 0000 7c00 0000  ..:...:.....|...
0x004000d0  0e00 0000 3c07 003f 9145 8460 1007 0060  ....<..?.E.`...`
0x004000e0  0ffe ab66 80b1 f1d6 6d3c 8016 4d31 d7e6  ...f....m<..M1..
0x004000f0  7d11 15a0 edd7 2188 e817 32e6 c7d8 05be  }.....!...2.....
0x00400100  67c4 77cd 56bc 9af4 3dee 2e85 10f4 44ee  g.w.V...=.....D.
0x00400110  2bab 45f6 a955 0456 5177 5211 269b 4ef5  +.E..U.VQwR.&.N.
0x00400120  5245 53f0 055e be85 7aa3 8d65 c966 6e09  RES..^..z..e.fn.
0x00400130  a098 2f3c e4ef 7fc7 5116 9083 c947 ad95  ../<....Q....G..
```

Non Unpackable Packer

The sample is an ITW threat, and goes with this ELF headers (ELFDump)

```
elf header:
    e_ident: ELFCLASS64 ELFDATA2LSB ELFOSABI_LINUX
    e_type: ET_EXEC
    e_machine: EM_X86_64
    e_version: EV_CURRENT
    e_entry: 0x55e148
    e_phoff: 64
    e_shoff: 0
    e_flags: 0
    e_ehsize: 64
    e_phentsize: 56
    e_phnum: 2
    e_shentsize: 64
    e_shnum: 0
    e_shstrndx: 0
```

```
entry: 0
    p_type: PT_LOAD
    p_offset: 0
    p_vaddr: 0x400000
    p_paddr: 0x400000
    p_filesz: 1438238
    p_memsz: 1438238
    p_flags: PF_X|PF_R
    p_align: 2097152
entry: 1
    p_type: PT_LOAD
    p_offset: 1799624
    p_vaddr: 0x9b75c8
    p_paddr: 0x9b75c8
    p_filesz: 0
    p_memsz: 0
    p_flags: PF_W|PF_R
    p_align: 2097152
```

Non Unpackable Packer

The sample is an ITW threat, and goes with this ELF headers (ReadELF):

```
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 03 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - GNU
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x55e148
  Start of program headers:          64 (bytes into file)
  Start of section headers:          0 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         2
  Size of section headers:           64 (bytes)
  Number of section headers:         0
  Section header string table index: 0

There are no sections in this file.

There are no sections to group in this file.
```

Non Unpackable Packer

The sample is an ITW threat, and goes with this ELF headers (ObjDump)

```
There are no sections to group in this file.

Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  LOAD           0x0000000000000000 0x0000000000400000 0x0000000000400000
                 0x000000000015f21e 0x000000000015f21e  R E    200000
  LOAD           0x000000000001b75c8 0x000000000009b75c8 0x000000000009b75c8
                 0x0000000000000000 0x0000000000000000  RW     200000

There is no dynamic section in this file.

There are no relocations in this file.

There are no unwind sections in this file.

Dynamic symbol information is not available for displaying symbols.

No version information found in this file.
```

Non Unpackable Packer

No strings what-so-ever... (r2 ; px, you can check further with iz?? too)

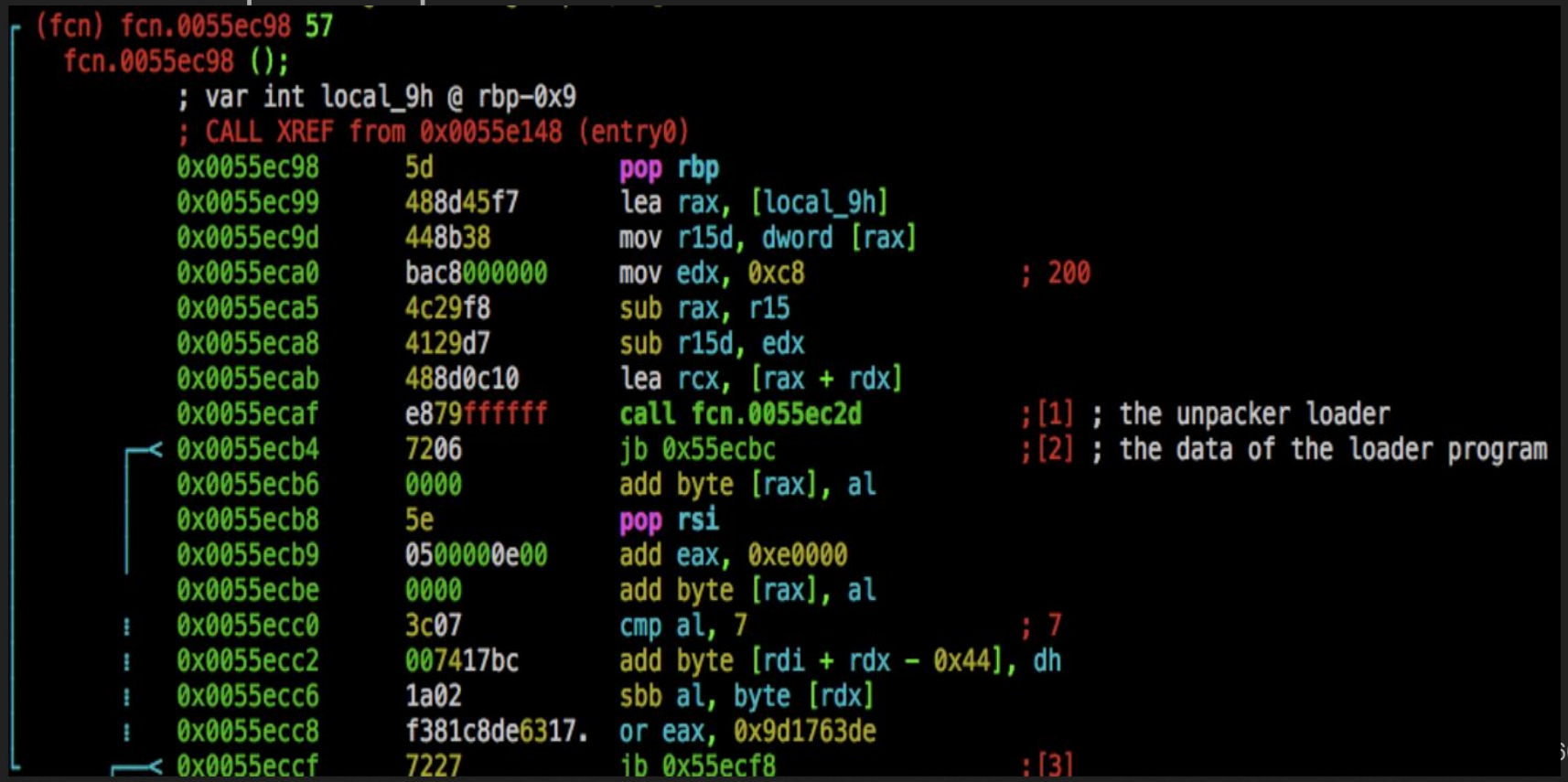
Non Unpackable Packer

This is how the entry point goes… see that first call

```
[0x0055e148]> af;pd 66
           ;-- rip:
 (fcn) entry0 2708
   entry0 (int arg_1h, int arg_10h);
           ; var int local_8h @ rbp-0x8
           ; arg int arg_1h @ rbp+0x1
           ; arg int arg_10h @ rbp+0x10
           ; var int local_38h @ rsp+0x38
           0x0055e148      e84b0b0000      call 0x55ec98 ; this is where the unpackingfunc is called
           0x0055e14d      55              push rbp
           0x0055e14e      53              push rbx
           0x0055e14f      51              push rcx
           0x0055e150      52              push rdx
           0x0055e151      4801fe          add rsi, rdi  ; '''
           0x0055e154      56              push rsi
```

Non Unpackable Packer

This what I call it as “unpacking’s loader” part, called by entry_0 1st call,
In here after the “loader program” is being executed, this unpacker program is used to decompress a packed of ELF data.



Non Unpackable Packer

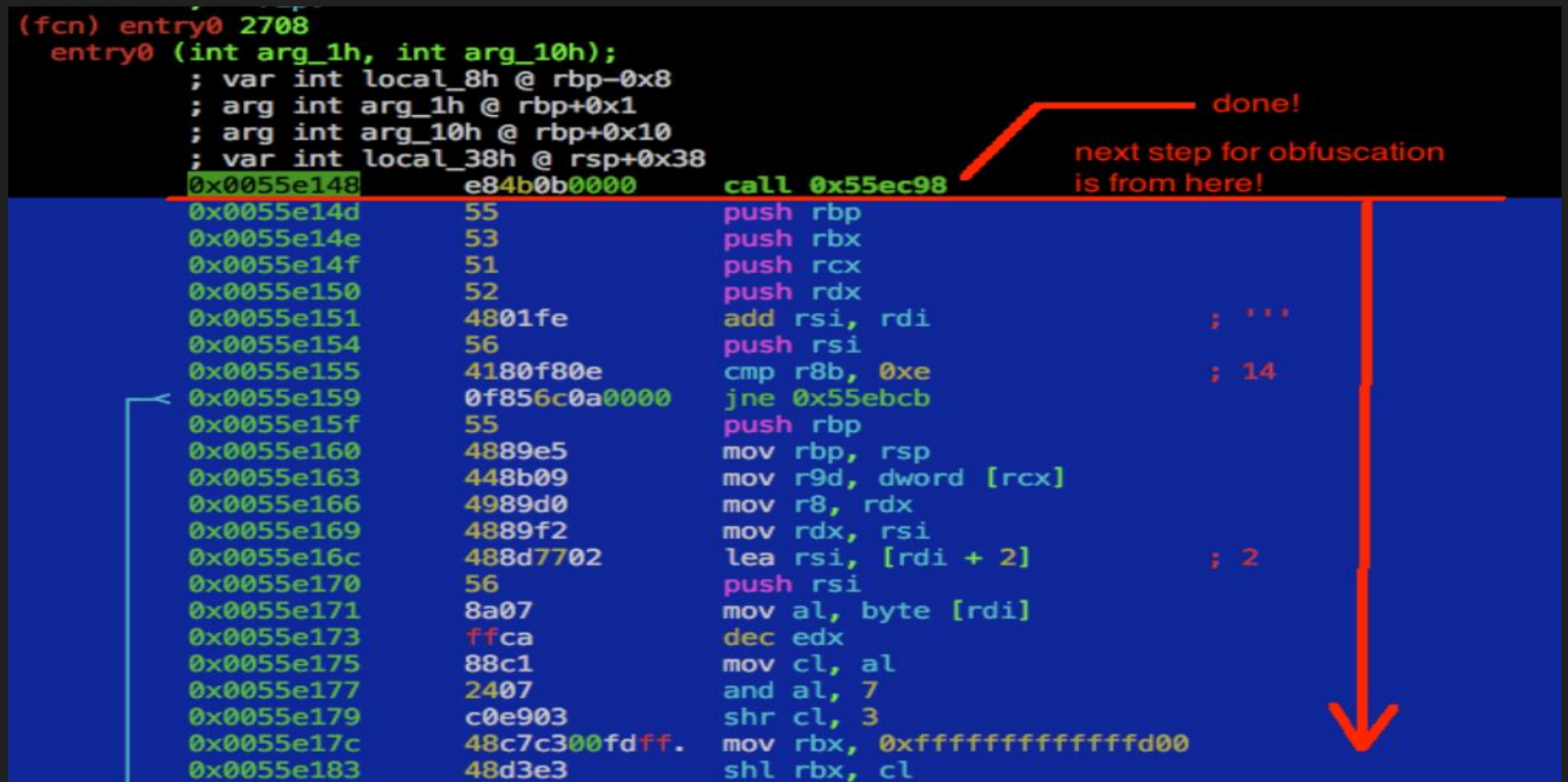
The loader logic load the packed data from the start of *0x40000c*, use reserved address *0x5bbf1a* as work-space & write + exec compress ELF result to base addr *0x400000*. (see the picture)

This loader has specific calls sequence: mmap, load.., exec, write, exit..

```
fcn.0055ec2d ();
     :    ; var int local_8h @ rsp+0x8
     :    ; var int local_18h @ rsp+0x18
     :    ; CALL XREF from 0x0055ecaf (fcn.0055ec98)
     :    0x0055ec2d      5b              pop rbx
     :    0x0055ec2e      6a01            push 1                          ; 1
     :    0x0055ec30      680c004000      push 0x40000c
     :    0x0055ec35      50              push rax
     :    0x0055ec36      681abf5b00      push 0x5bbf1a
     :    0x0055ec3b      51              push rcx
     :    0x0055ec3c      4157            push r15
     :    0x0055ec3e      bf0000a000      mov edi, 0xa00000
     :    0x0055ec43      6a07            push 7                          ; 7
     :    0x0055ec45      5a              pop rdx
     :    0x0055ec46      be1abf5b00      mov esi, 0x5bbf1a
     :    0x0055ec4b      6a32            push 0x32                       ; '2' ; 50
     :    0x0055ec4d      415a            pop r10
     :    0x0055ec4f      4529c0          sub r8d, r8d
     :    0x0055ec52      6a09            push 9                          ; 9
     :    0x0055ec54      58              pop rax
     :    0x0055ec55      0f05            syscall
     :    0x0055ec57      39c7            cmp edi, eax
     `=<  0x0055ec59      0f857dffffff    jne 0x55ebdc
          0x0055ec5f      be00004000      mov esi, 0x400000               ; section.ehdr
          0x0055ec64      89fa            mov edx, edi
          0x0055ec66      29f2            sub edx, esi
     ,=<  0x0055ec68      7415            je 0x55ec7f
     |    0x0055ec6a      01d5            add ebp, edx
     |    0x0055ec6c      01542408        add dword [local_8h], edx
     |    0x0055ec70      01542418        add dword [local_18h], edx
     |    0x0055ec74      89d9            mov ecx, ebx
     |    0x0055ec76      29f1            sub ecx, esi
     |    0x0055ec78      c1e903          shr ecx, 3
     |    0x0055ec7b      fc              cld
     |    0x0055ec7c      f348a5          rep movsq qword [rdi], qword ptr [rsi]
     `->  0x0055ec7f      97              xchg eax, edi
          0x0055ec80      4889de          mov rsi, rbx
          0x0055ec83      50              push rax
```

Non Unpackable Packer

After successfully decompressed the packed data (on memory), the packed data is still in obfuscated form. The next process in entry point will be used to load the image & deobfuscate it (blue marked region)..

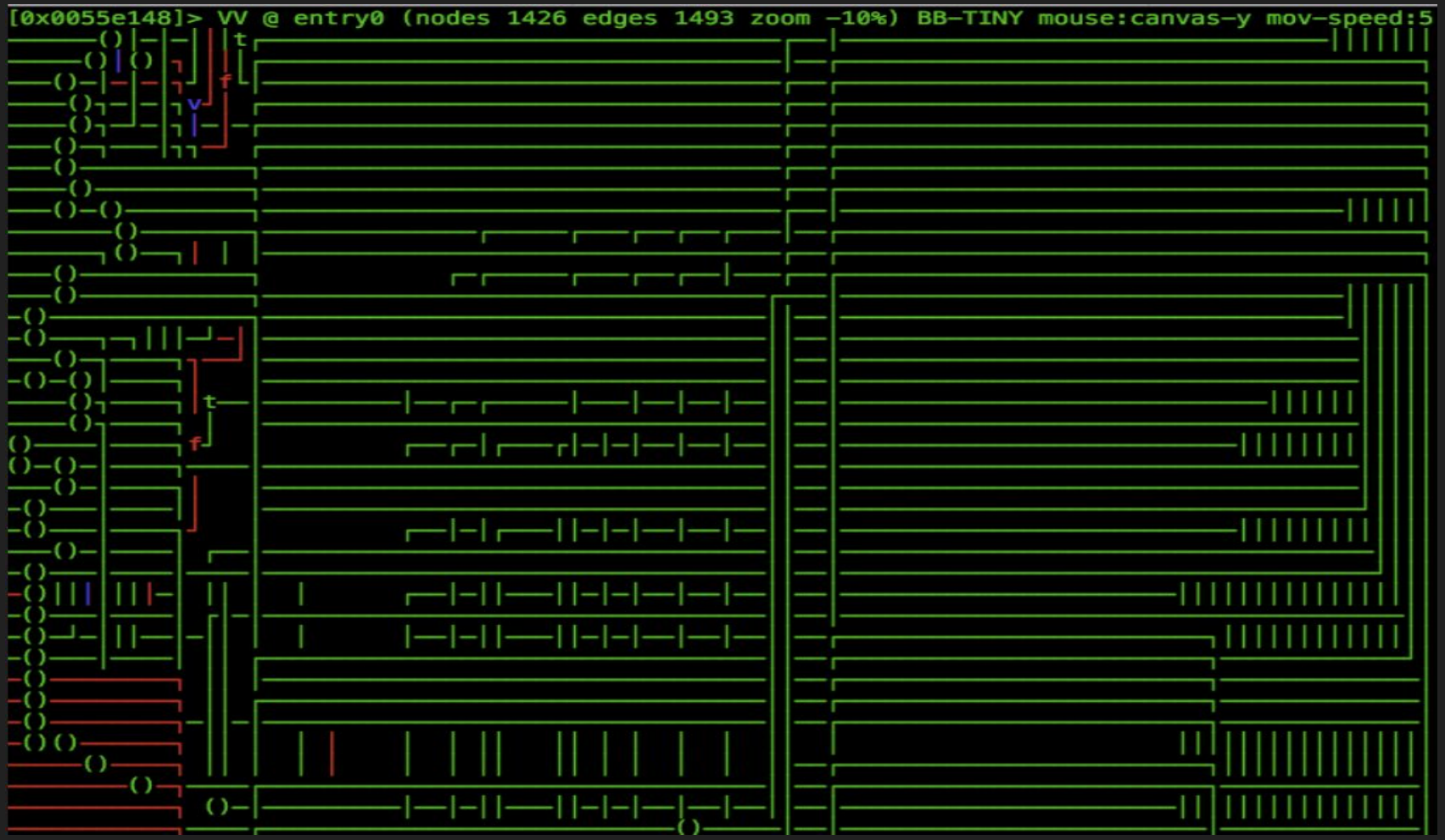


Non Unpackable Packer

At the described on the blue section, the loaded packed data will be deobfuscated on several loop of subroutines looks like this in radare2:

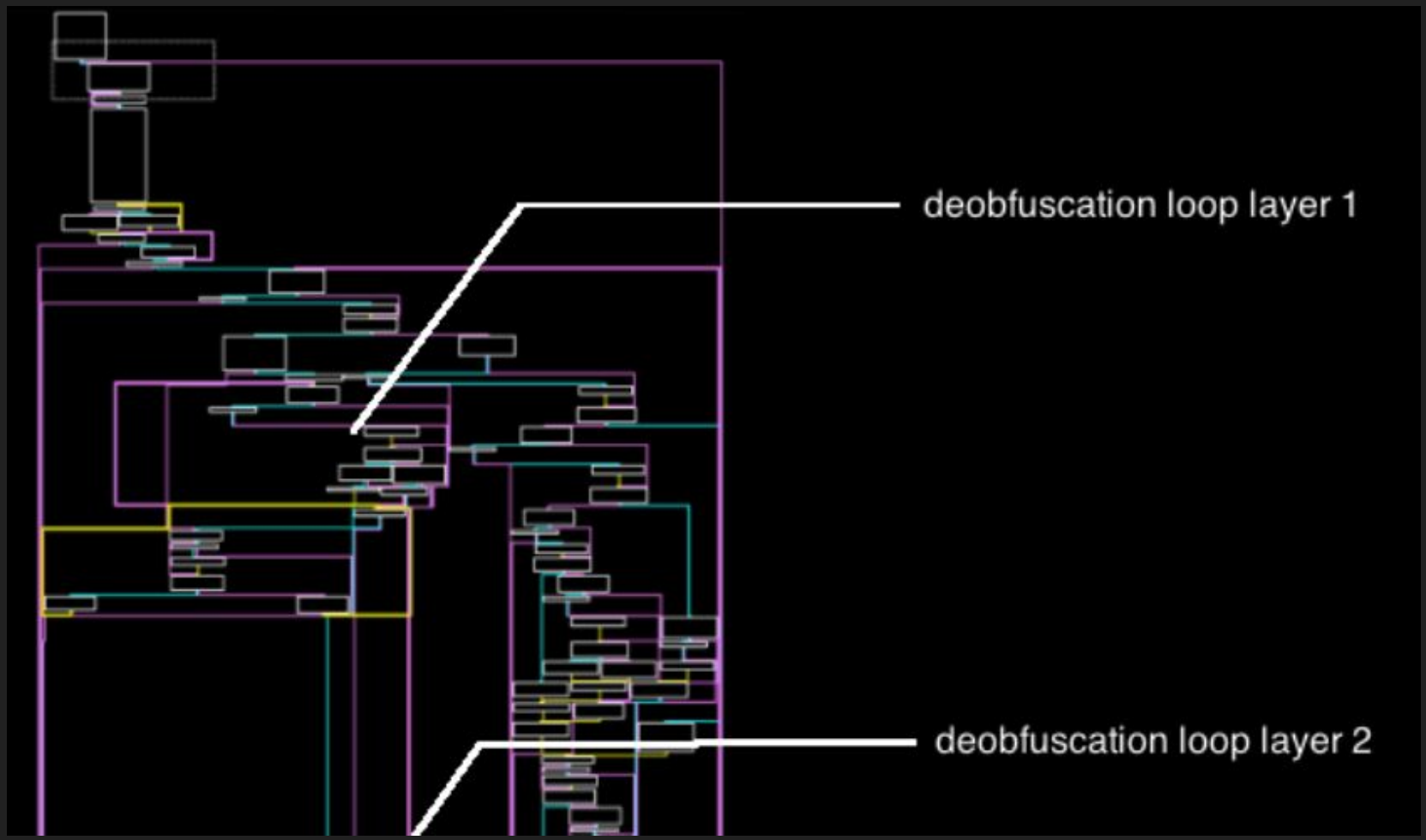
Non Unpackable Packer

I can not see much of it in the normal mode.. will need a super big screen..



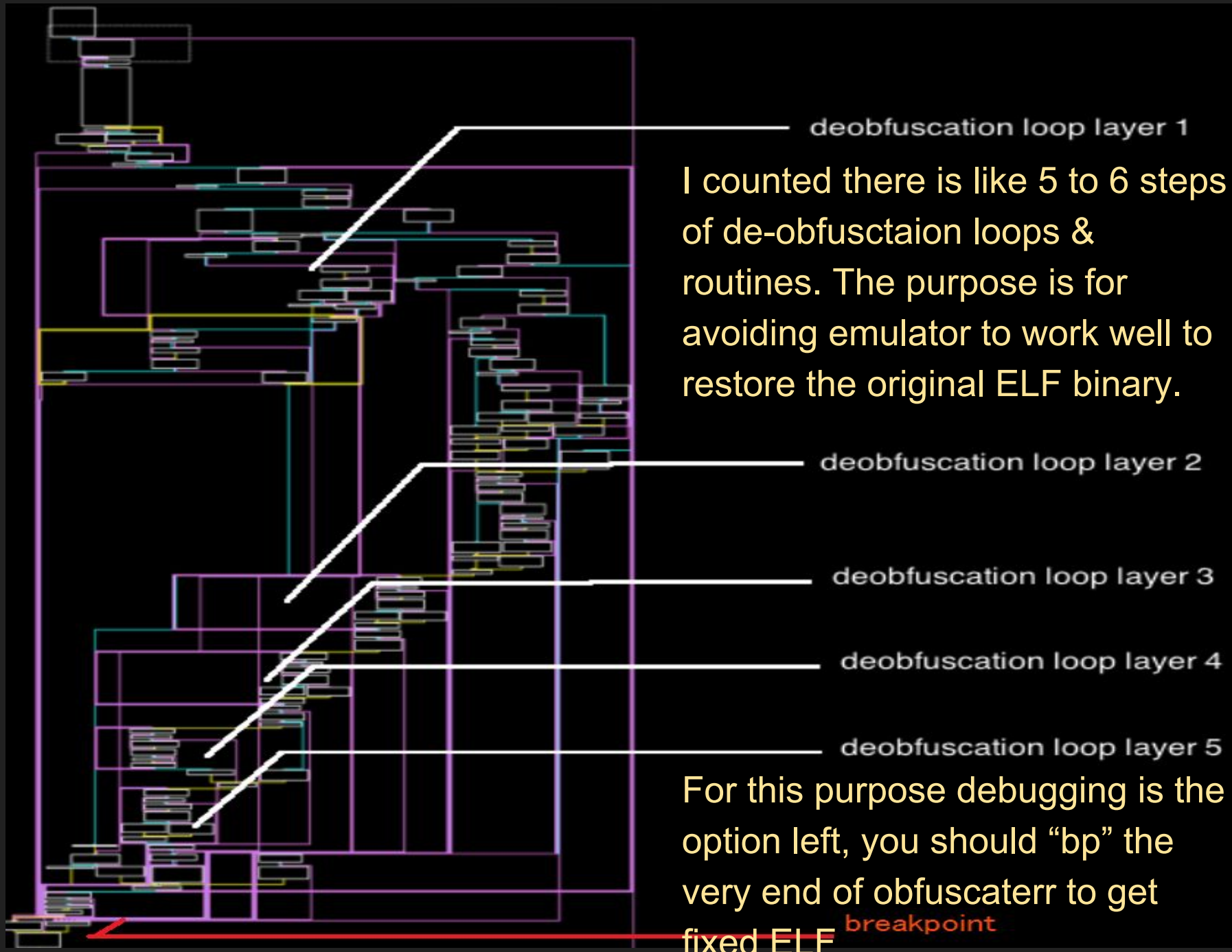
Non Unpackable Packer

Borrowing some blah blah tool graph, It actually looks like this….(part one)



Non Unpackable Packer

It actually looks like this….(part two)

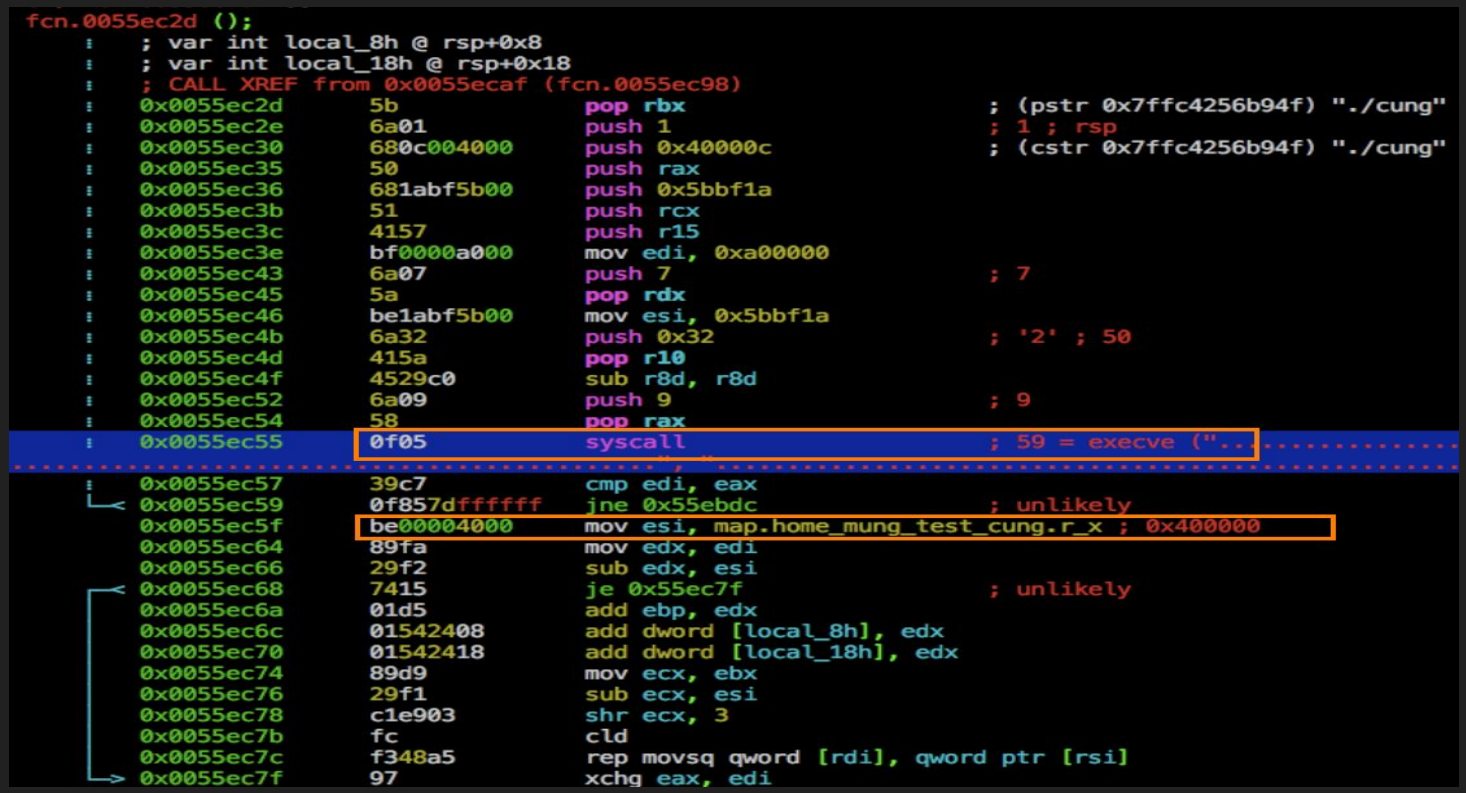


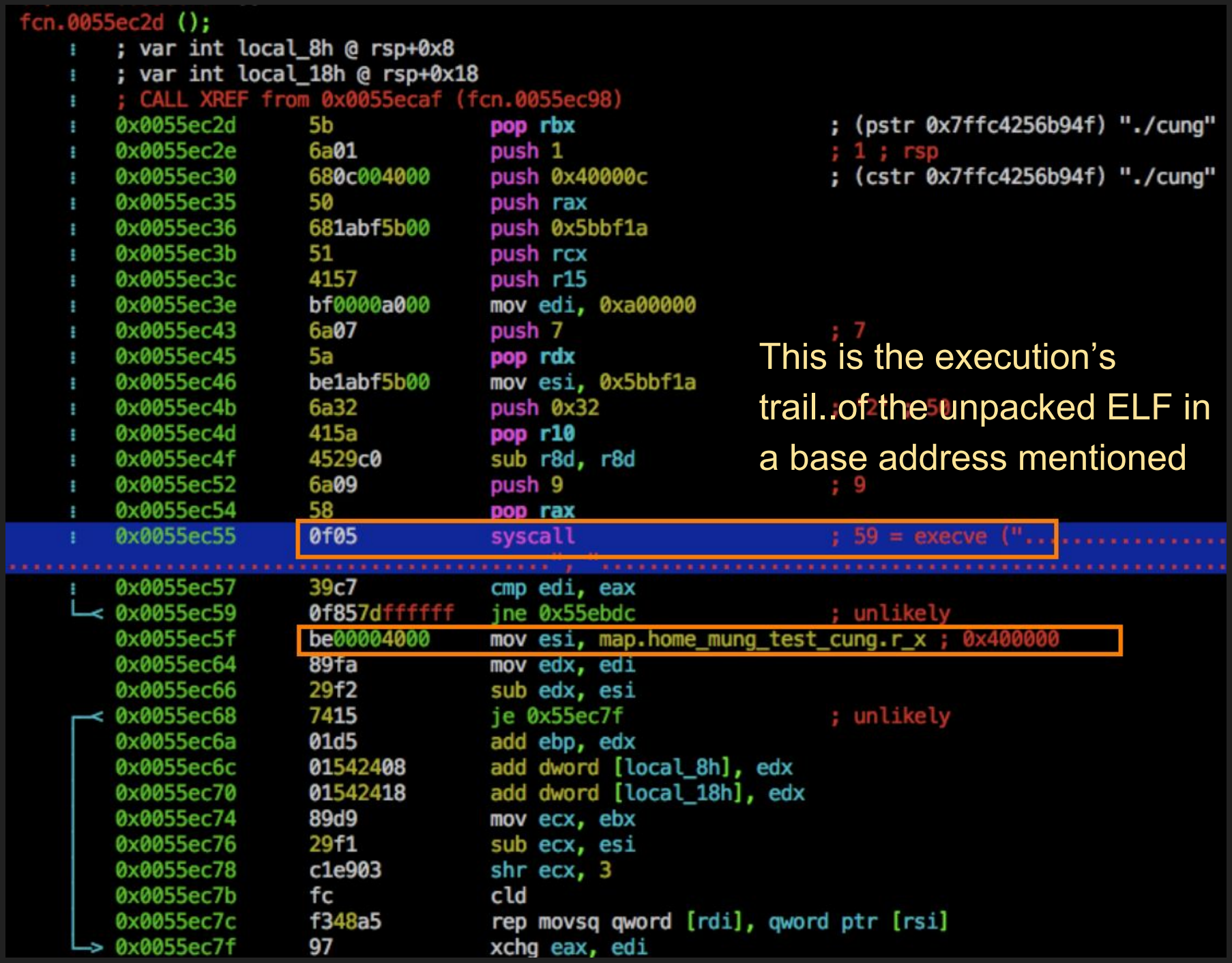
I counted there is like 5 to 6 steps of de-obfusctaion loops & routines. The purpose is for avoiding emulator to work well to restore the original ELF binary.

For this purpose debugging is the option left, you should “bp” the very end of obfuscaterr to get fixed ELF

Non Unpackable Packer

Unpacked binary is written & exec at base address *0x400000* w/same image name. You can use ESIL to see the this process better, but don't use ESIL to extract the packed ELF since 1 byte mistake will ruin real ELF





This is the execution's trail..of the unpacked ELF in a base address mentioned

Non Unpackable Packer

Other tip is, same method of analysis can be done by command pdda using r2dec (great work! > r2dec @wagio).

```
0x0055ec16  and byte [rax], ah                  |   *(rax) &= ah;
0x0055ec18  and byte [rdx], cl                  |   *(rdx) &= cl;
0x0055ec1a  add byte [rax + 0x5f026a5e], dl     |   *(rax + 0x5f026a5e) += dl;
0x0055ec20  push 1                              |
0x0055ec22  pop rax                             |   rax = 1;
0x0055ec23  syscall                             |   __asm (syscall);
0x0055ec25  push 0x7f                           |
0x0055ec27  pop rdi                             |   rdi = 0x7f;
0x0055ec28  push 0x3c                           |
0x0055ec2a  pop rax                             |   rax = 0x3c;
0x0055ec2b  syscall                             |   __asm (syscall);
0x0055ec2d  pop rbx                             |
0x0055ec2e  push 1                              |
0x0055ec30  push 0x40000c                       |
0x0055ec35  push rax                            |
0x0055ec36  push 0x5bbf1a                       |
0x0055ec3b  push rcx                            |
0x0055ec3c  push r15                            |
0x0055ec3e  mov edi, 0xa00000                   |   edi = 0xa00000;
0x0055ec43  push 7                              |
0x0055ec45  pop rdx                             |   rdx = 7;
0x0055ec46  mov esi, 0x5bbf1a                   |   esi = 0x5bbf1a;
0x0055ec4b  push 0x32                           |
0x0055ec4d  pop r10                             |   r10 = 0x32;
0x0055ec4f  sub r8d, r8d                        |   r8d -= r8d;
0x0055ec52  push 9                              |
0x0055ec54  pop rax                             |   rax = 9;
0x0055ec55  syscall                             |   __asm (syscall);
0x0055ec57  cmp edi, eax                        |
0x0055ec59  invalid                             |
0x0055ec5a  invalid                             |
                                                |   if (edi < eax) {
0x0055ec5b  jge 0x55ec5c                        |
                                                |   }
0x0055ec5d  invalid                             |
0x0055ec5e  invalid                             |
0x0055ec5f  mov esi, 0x400000                   |   esi = map.home_mung_test_cung.r_x;
                                                | }
```

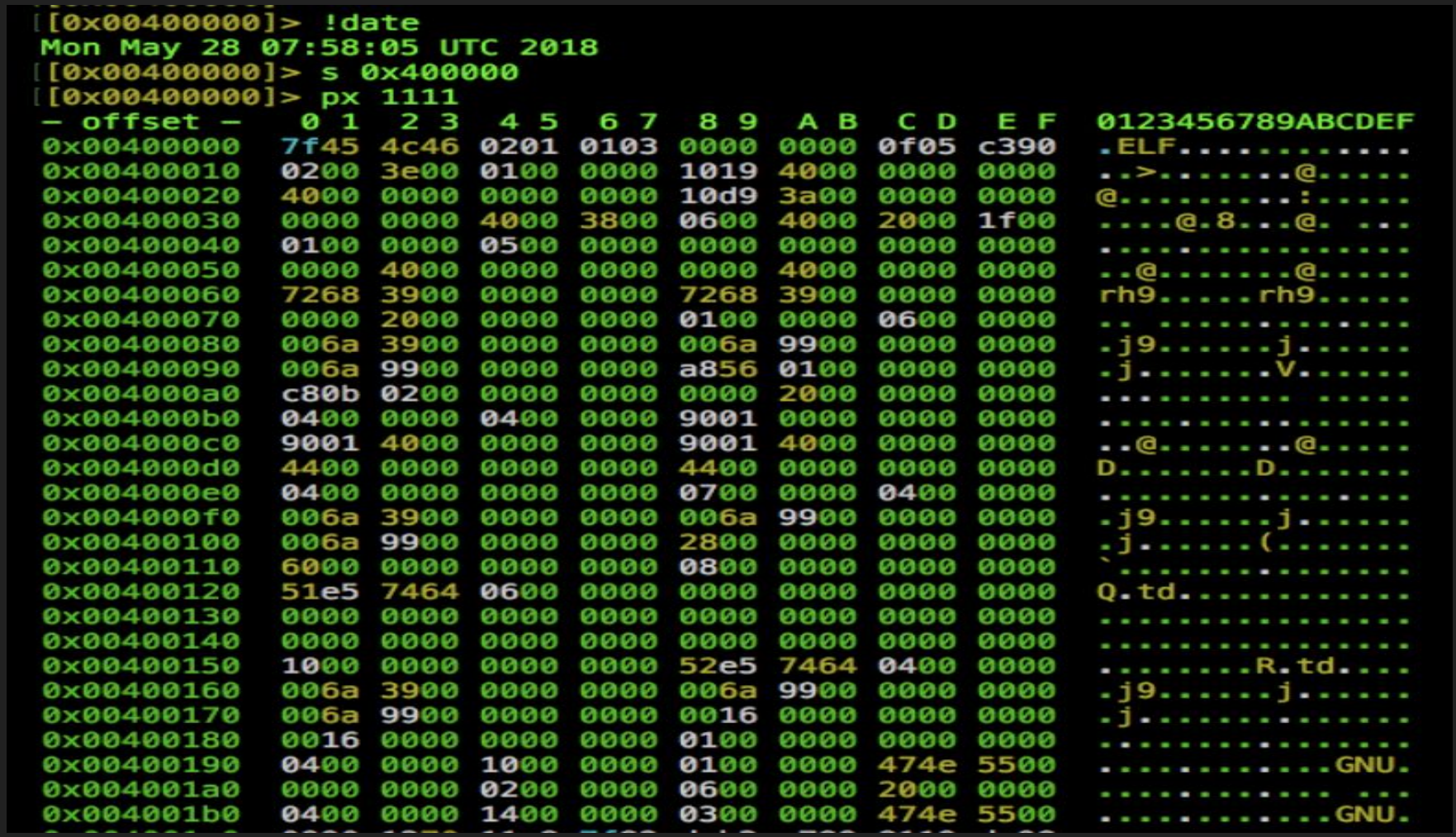
```
0x0055ec16  and byte [rax], ah                |          *(rax) &= ah;
0x0055ec18  and byte [rdx], cl                |          *(rdx) &= cl;
0x0055ec1a  add byte [rax + 0x5f026a5e], dl   |          *(rax + 0x5f026a5e) += dl;
0x0055ec20  push 1                            |
0x0055ec22  pop rax                           |          rax = 1;
0x0055ec23  syscall                           |          __asm (syscall);
0x0055ec25  push 0x7f                         |
0x0055ec27  pop rdi                           |          rdi = 0x7f;
0x0055ec28  push 0x3c                         |
0x0055ec2a  pop rax                           |          rax = 0x3c;
0x0055ec2b  syscall                           |          __asm (syscall);
0x0055ec2d  pop rbx                           |
0x0055ec2e  push 1                            |
0x0055ec30  push 0x40000c                     |
0x0055ec35  push rax                          |
0x0055ec36  push 0x5bbf1a                     |
0x0055ec3b  push rcx                          |
0x0055ec3c  push r15                          |
0x0055ec3e  mov edi, 0xa00000                 |          edi = 0xa00000;
0x0055ec43  push 7                            |
0x0055ec45  pop rdx                           |          rdx = 7;
0x0055ec46  mov esi, 0x5bbf1a                 |          esi = 0x5bbf1a;
0x0055ec4b  push 0x32                         |
0x0055ec4d  pop r10                           |          r10 = 0x32;
0x0055ec4f  sub r8d, r8d                      |          r8d -= r8d;
0x0055ec52  push 9                            |
0x0055ec54  pop rax                           |          rax = 9;
0x0055ec55  syscall                           |          __asm (syscall);
0x0055ec57  cmp edi, eax                      |
0x0055ec59  invalid                           |
0x0055ec5a  invalid                           |
                                              |          if (edi < eax) {
0x0055ec5b  jge 0x55ec5c                      |
                                              |          }
0x0055ec5d  invalid                           |
0x0055ec5e  invalid                           |
0x0055ec5f  mov esi, 0x400000                 |          esi = map.home_mung_test_cung.r_x;
                                              |      }
```

Way to go Giovanni!

R2dec nailed it good!

Non Unpackable Packer

The successfully unpacked binary is being rewritten at *0x400000* like this.



Non Unpackable Packer

After the unpacked ELF formed, we can dump it (I use “wtf”) and check:

```
33594 0x002a0aac 0x006a0aac   8   9 (LOAD0) ascii CRAM-MD5
33595 0x002a0ab5 0x006a0ab5   4   5 (LOAD0) ascii AQ==
33596 0x002a0aba 0x006a0aba   6   7 (LOAD0) ascii GSSAPI
33597 0x002a0ac8 0x006a0ac8  41  42 (LOAD0) ascii Unsupported SASL authentication mechanism
33598 0x002a0b38 0x006a0b38   4   5 (LOAD0) ascii /../
33599 0x002a0b3d 0x006a0b3d  20  21 (LOAD0) ascii Invalid input packet
33600 0x002a0b52 0x006a0b52   4   5 (LOAD0) ascii SMBS
33601 0x002a0b7c 0x006a0b7c  10  11 (LOAD0) ascii NT LM 0.12
33602 0x002a0b88 0x006a0b88  67  68 (LOAD0) ascii %s %02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x
33603 0x002a0bcc 0x006a0bcc   7   8 (LOAD0) ascii nonce="
33604 0x002a0bd4 0x006a0bd4   7   8 (LOAD0) ascii realm="
33605 0x002a0bdc 0x006a0bdc  10  11 (LOAD0) ascii algorithm=
33606 0x002a0be7 0x006a0be7   5   6 (LOAD0) ascii qop="
33607 0x002a0bed 0x006a0bed   8   9 (LOAD0) ascii md5-sess
33608 0x002a0bf6 0x006a0bf6   8   9 (LOAD0) ascii auth-int
33609 0x002a0bff 0x006a0bff   9  10 (LOAD0) ascii auth-conf
33610 0x002a0c09 0x006a0c09  16  17 (LOAD0) ascii %08x%08x%08x%08x
33611 0x002a0c1a 0x006a0c1a  19  20 (LOAD0) ascii %s:%s:%08x:%s:%s:%s
33612 0x002a0c2e 0x006a0c2e  15  16 (LOAD0) ascii %s, opaque="%s"
33613 0x002a0c3e 0x006a0c3e  18  19 (LOAD0) ascii %s, algorithm="%s"
33614 0x002a0c51 0x006a0c51   5   6 (LOAD0) ascii stale
33615 0x002a0c57 0x006a0c57   5   6 (LOAD0) ascii realm
33616 0x002a0c5d 0x006a0c5d   6   7 (LOAD0) ascii opaque
33617 0x002a0c68 0x006a0c68   8   9 (LOAD0) ascii MD5-sess
33618 0x002a0c78 0x006a0c78  90  91 (LOAD0) ascii username="%s",realm="%s",nonce="%s",cnonce="%s",nc="%s",digest-uri="%s",response=%s,
33619 0x002a0cd8 0x006a0cd8  32  33 (LOAD0) ascii d41d8cd98f00b204e9800998ecf8427e
33620 0x002a0d00 0x006a0d00  92  93 (LOAD0) ascii username="%s", realm="%s", nonce="%s", uri="%s", cnonce="%s", nc=%08x, qop=%s, resp
33621 0x002a0d60 0x006a0d60  62  63 (LOAD0) ascii username="%s", realm="%s", nonce="%s", uri="%s", response="%s"
33622 0x002a0da0 0x006a0da0  46  47 (LOAD0) ascii NTLM handshake failure (empty type-2 message)\n
33623 0x002a0dd0 0x006a0dd0  44  45 (LOAD0) ascii NTLM handshake failure (bad type-2 message)\n
33624 0x002a0e00 0x006a0e00  97  98 (LOAD0) ascii NTLM handshake failure (bad type-2 message). Target Info Offset Len is set incorrec
33625 0x002a0e68 0x006a0e68   9  10 (LOAD0) ascii NTLMSSP%c
33626 0x002a0e72 0x006a0e72  50  51 (LOAD0) ascii %c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%s%s
33627 0x002a0ea8 0x006a0ea8  42  43 (LOAD0) ascii gethostname() failed, continuing without!\n
33628 0x002a0ed8 0x006a0ed8   9  10 (LOAD0) ascii NTLMSSP%c
33629 0x002a0ee2 0x006a0ee2 110 111 (LOAD0) ascii %c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%
%c%c%c%c%c
```

Non Unpackable Packer

Fix the address correctly with base 0x400000 and you can analyze the ELF as usual static ways to ID it, in example, like this, boom! Busted!

```
[0x00400000]> / var
Searching 3 bytes in [0x400000-0x797000]
hits: 15
0x00402b93 hit1_0 .fH/var/run$ H$.
0x00402bb3 hit1_1 .$ .H/var/lockH$ .
0x00402bf3 hit1_2 .$ H/var/tmp$ H$.
0x0064cfa9 hit1_3 .AIF*H/var/runH`f`.
0x00696127 hit1_4 . out/dev/shm//var/tmp//var/lock/.
0x00696131 hit1_5 .shm//var/tmp//var/lock//var/run/.
0x0069613c hit1_6 .mp//var/lock//var/run/%s/%s.xmr.
0x006a78c0 hit1_7 .Godavari X4.
0x00708750 hit1_8 .own module namevariable has no val.
0x0070886f hit1_9 . or environment variableconf_mod.c.
0x00711834 hit1_10 . many temporary variablesBNRANDBN.
0x00716d8a hit1_11 .d_funcDSO_bind_varDSO_convert_fil.
0x00717b30 hit1_12 .EAY_RAND_BYTES/var/run/egd-pool/d.
0x00730419 hit1_13 .e8e/var/tmp/var/profil.
0x00730422 hit1_14 .e/var/tmp/var/profilee.
[0x00400000]> / xmr
Searching 3 bytes in [0x400000-0x797000]
hits: 3
0x0069614c hit2_0 .var/run/%s/%s.xmrt[+] Already ru.
0x006966c8 hit2_1 .optimizations.xmrt 0.1 built on .
0x00699523 hit2_2 .1;32m * [01;37mxmrt 0.1    [01;36m.
[0x00400000]> / Tiny
Searching 4 bytes in [0x400000-0x797000]
hits: 1
0x006963e2 hit3_0 .ation.+ Tiny XMR mooner.++.
```

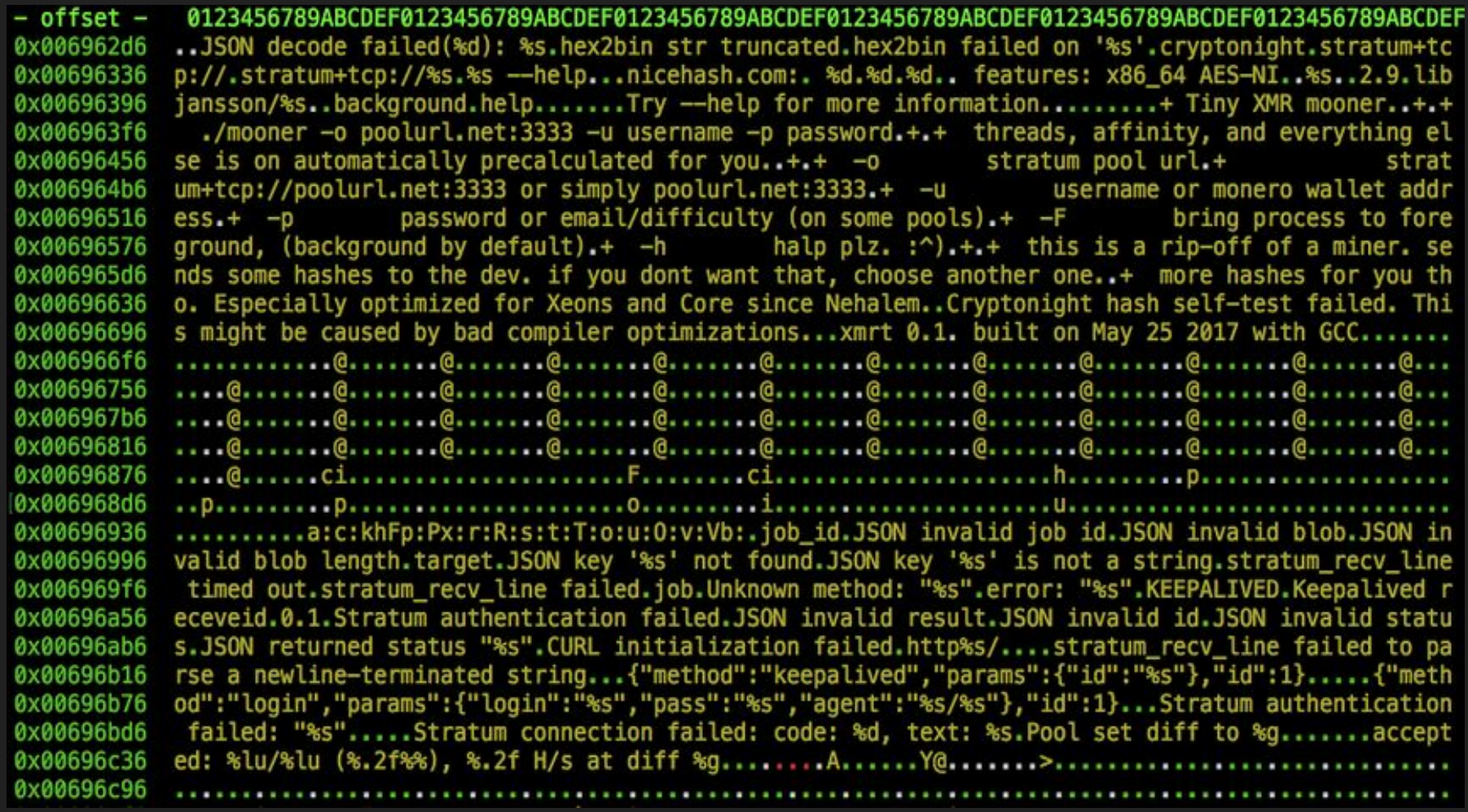
Non Unpackable Packer

PS: This ELF is also having own self decryption function (encrypted), but it can be handled by usual analysis.. You can hack this statically but as I said I am a debugger, so..I debug stuff faster.

```
[0x00401910]> i~file
file     dump-bin-elf
type     EXEC (Executable file)
[0x00401910]> i~mac
machine  AMD x86-64 architecture
[0x00401910]>
[0x00401910]>
[0x00401910]> af:pdf
[0x00401910]> af;pdf
           ;-- entry0:
 (fcn) fcn.rip 42
   fcn.rip ();
           0x00401910      31ed           xor ebp, ebp
           0x00401912      4989d1         mov r9, rdx
           0x00401915      5e             pop rsi
           0x00401916      4889e2         mov rdx, rsp
           0x00401919      4883e4f0       and rsp, 0xfffffffffffffff0
           0x0040191d      50             push rax
           0x0040191e      54             push rsp
           0x0040191f      49c7c0801a5e.  mov r8, 0x5e1a80  ; this is the decrypter 1
           0x00401926      48c7c1f0195e.  mov rcx, 0x5e19f0 ; this is the decrypter 2, to then exec the main
           0x0040192d      48c7c7a01340.  mov rdi, 0x4013a0 ; main ; "AVAU1\xc0ATUH\x89\xf5S\x89\xfb\xe8\x9c\x16
           0x00401934      e857f81d00     call fcn.005e1190
           0x00401939      f4             hlt
[0x00401910]>
```

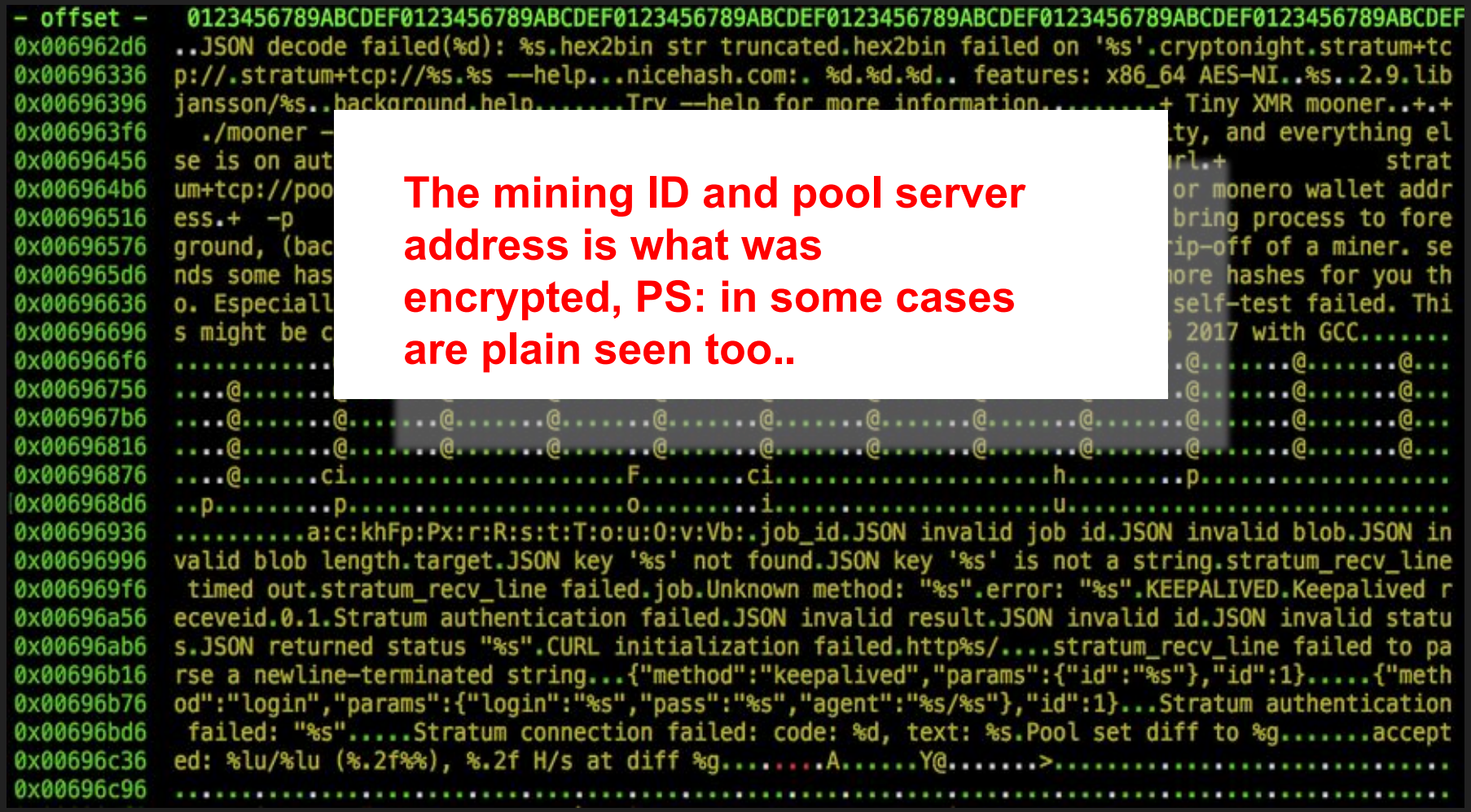
Non Unpackable Packer

..you can analyze the packed ELF process too in memory, to check stuff:



Non Unpackable Packer

To make it short, this is what was encrypted.. (gained by debug mode)



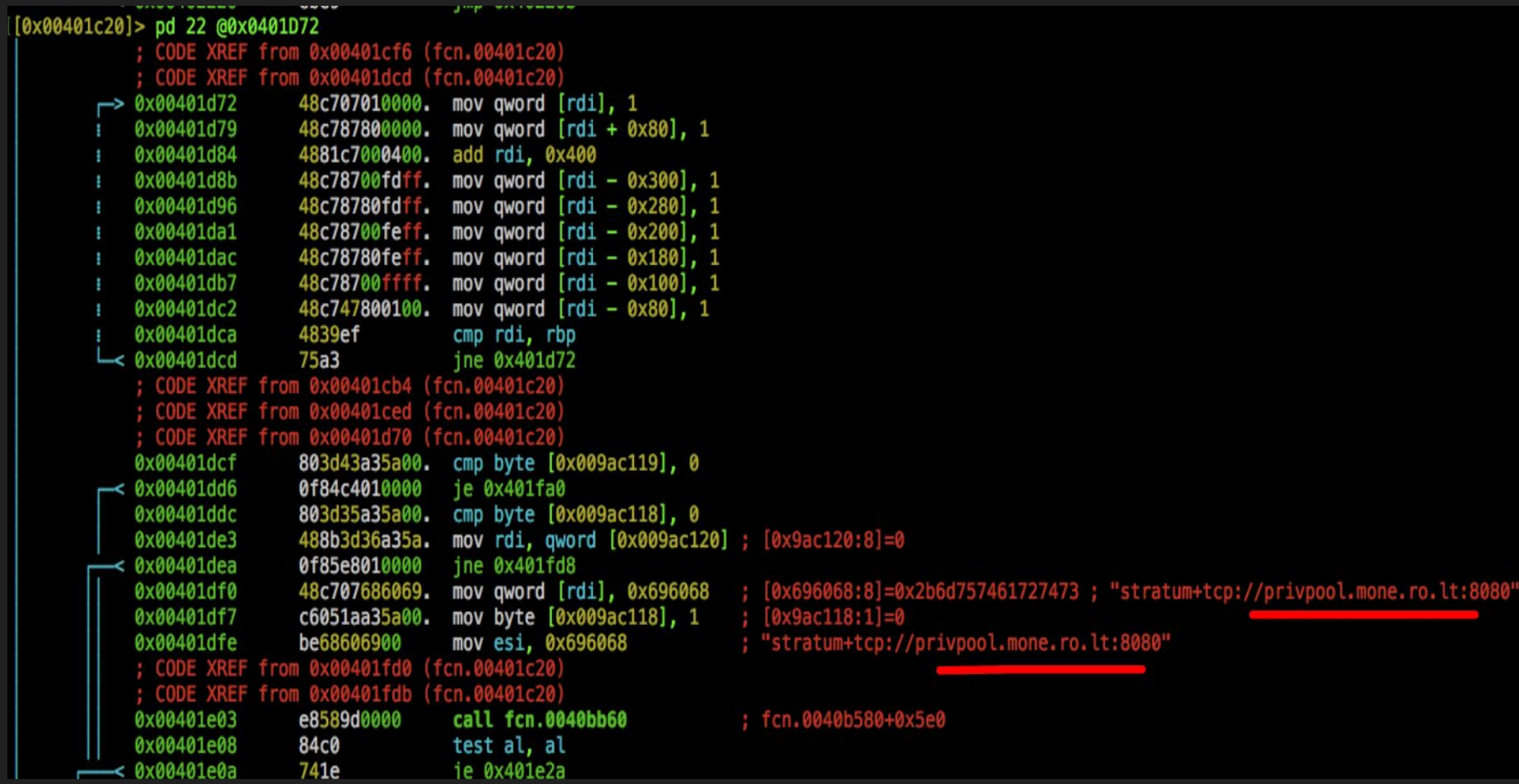
Non Unpackable Packer

This ELF is actually letting user mining with its own pre-set configuration like pool server, user id/password set into "Donation pool" mode function if the mining config data is not provided… (of course that config wasn't even executed in command line or config JSON file...) Hmm.. weird..

```
0x00401e6e      4183f201        xor r10d, 1
0x00401e72      80fb01          cmp bl, 1                       ; 1
0x00401e75      4c0f440dd3a2.   cmove r9, qword [0x009ac150]
0x00401e7d      44881596a25a.   mov byte [0x009ac11a], r10b ; [0x9ac11a:1]=0
0x00401e84      4531e4          xor r12d, r12d
0x00401e87      4d890b          mov qword [r11], r9
0x00401e8a      e871572300      call fcn.00637600
0x00401e8f      beaf606900      mov esi, 0x6960af               ; "Switching to donation pool"
0x00401e94      bf01000000      mov edi, 1
0x00401e99      31c0            xor eax, eax
0x00401e9b      e8c0c80100      call fcn.0041e760
```

Non Unpackable Packer

Binaries just don’t lie..



Non Unpackable Packer

Boom! Your carefully hidden miner servers has just been pwned..

```
cmp byte [0x009ac119], 0
je 0x401fa0
cmp byte [0x009ac118], 0
mov rdi, qword [0x009ac120] ; [0x9ac120:8]=0
jne 0x401fd8
mov qword [rdi], 0x696068   ; [0x696068:8]=0x2b6d757461727473 ; "stratum+tcp://privpool.mone.ro.lt:8080"
mov byte [0x009ac118], 1    ; [0x9ac118:1]=0
mov esi, 0x696068           ; "stratum+tcp://privpool.mone.ro.lt:8080"
```

Non Unpackable Packer

Infection/installation artifacts to check:
==========
(not so much since the process are mostly on memory)

Checks for:

- /dev/shm/
- /var/tmp/
- /var/lock/
- /var/run/
- /sys/devices/system/cpu/online"

(others are not that significant, like libs, config or resolv or host or gai.conf, I don't mention here)

Writes:

- /var/lock/.xmrt (lock file)
- /var/run/.xmrt.pid (pid file)
- /tmp/.xmrt (temporary work file)

Networking calls:
---
1. socket(PF_NETLINK, SOCK_RAW, 0) and bind(4, {sa_family=AF_NETLINK, pid=0, groups=00000000}, 12) for IP lookup, NOTED: literally, "pid=0", "groups=0"
2. socket(PF_INET, SOCK_DGRAM, IPPROTO_IP) and connect(4, {sa_family=AF_INET, sin_port=htons(8080), sin_addr=inet_addr("xxx")}, 16) to then endto() and recvform() to the pool server(s) with cryptonite JSON.

Non Unpackable Packer

Q: How many packed binaries using such packers?

A: A lot, and various customization, for example, this one (see the first call)

```
MILEMINE.FIL: ELF 64-bit LSB executable, x86-64,
version 1 (GNU/Linux), statically linked, stripped
```

```
        ;-- entry0:
┌ (fcn) fcn.rip 50
│   fcn.rip ();
│           0x0045adf0      e8df010000      call fcn.0045afd4          ;[1] ; firstly loading the loader to unpack
│           0x0045adf5      55              push rbp
│           0x0045adf6      53              push rbx
│           0x0045adf7      51              push rcx
│           0x0045adf8      52              push rdx
│           0x0045adf9      4801fe          add rsi, rdi               ; '''
│           0x0045adfc      56              push rsi
│           0x0045adfd      4889fe          mov rsi, rdi
│           0x0045ae00      4889d7          mov rdi, rdx
│           0x0045ae03      31db            xor ebx, ebx
│           0x0045ae05      31c9            xor ecx, ecx
│           0x0045ae07      4883cdff        or rbp, 0xffffffffffffffff
│           0x0045ae0b      e850000000      call fcn.0045ae60          ;[2]
│           0x0045ae10      01db            add ebx, ebx
│       ┌<  0x0045ae12      7402            je 0x45ae16                ;[3]
│       │   0x0045ae14      f3c3            ret
│       └-> 0x0045ae16      8b1e            mov ebx, dword [rsi]
```

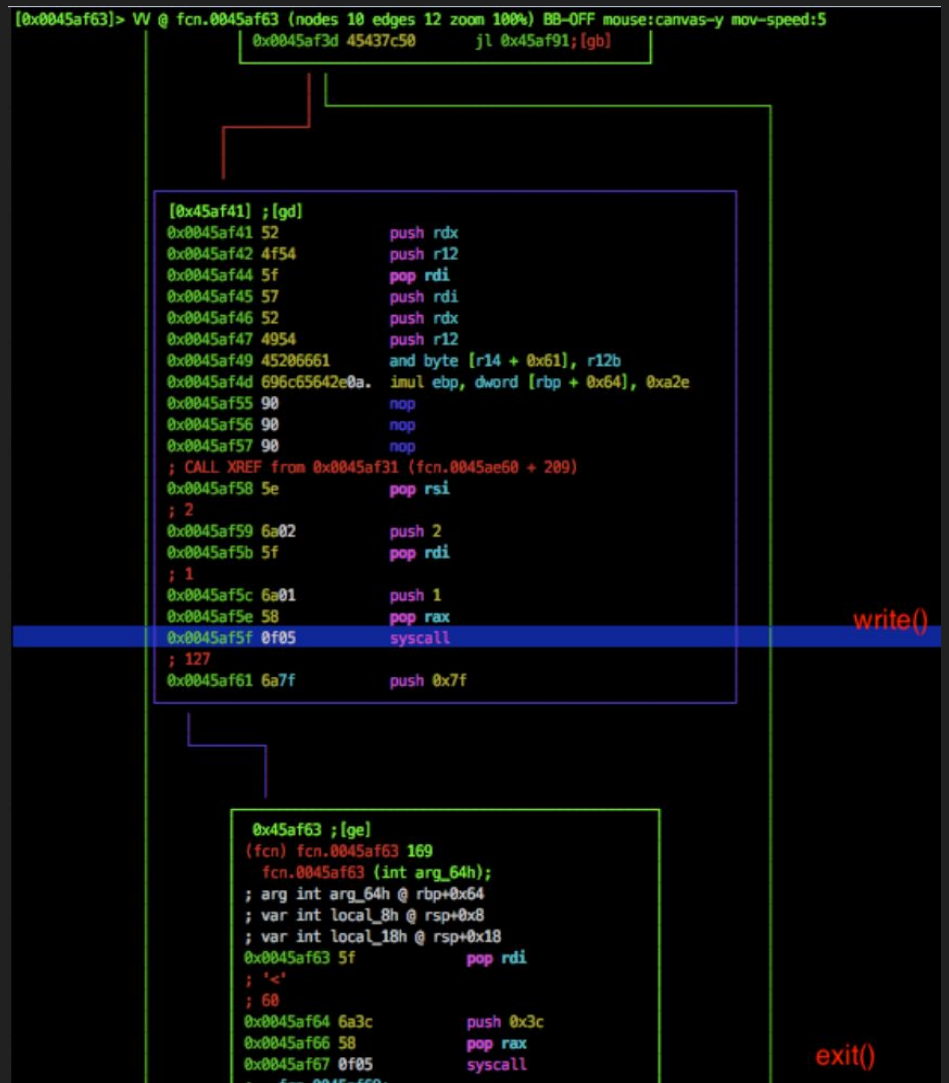
Non Unpackable Packer

The loader part always call to loader part contains loads a program & data..

```
(fcn) fcn.0045afd4 40
  fcn.0045afd4 ();
          ; var int local_9h @ rbp-0x9
          ; CALL XREF from 0x0045adf0 (fcn.rip)
          0x0045afd4      5d              pop rbp
          0x0045afd5      488d45f7        lea rax, [local_9h]
          0x0045afd9      448b38          mov r15d, dword [rax]
          0x0045afdc      bac8000000      mov edx, 0xc8               ; 200
          0x0045afe1      4c29f8          sub rax, r15
          0x0045afe4      4129d7          sub r15d, edx
          0x0045afe7      488d0c10        lea rcx, [rax + rdx]
          0x0045afeb      e879ffffff      call fcn.0045af69           ;[1] ; mmap then write and exec
      ┌<  0x0045aff0      7206            jb 0x45aff8                 ;[2] ; load binary data of the loader.
      │   0x0045aff2      0000            add byte [rax], al
      │   0x0045aff4      6c              insb byte [rdi], dx
      │   0x0045aff5      0500000849      add eax, 0x49080000
      │   0x0045affa      1100            adc dword [rax], eax
```

Non Unpackable Packer

Loader program itself, same steps: *mmap, load.., exec, write, exit*..



Non Unpackable Packer

The packed ELF data is in the same address, of *0x40000c*

```
[0x0045af69]> #the packed data is always in 0x40000c
[0x0045af69]>
[0x0045af69]> x @0x40000c!2000
- offset -   0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0123456789ABCDEF
0x0040000c  0000 0000 0200 3e00 0100 0000 f0ad 4500  ......>.......E.
0x0040001c  0000 0000 4000 0000 0000 0000 0000 0000  ....@...........
0x0040002c  0000 0000 0000 0000 4000 3800 0200 4000  ........@.8...@.
0x0040003c  0000 0000 0100 0000 0500 0000 0000 0000  ................
0x0040004c  0000 0000 0000 4000 0000 0000 0000 4000  ......@.......@.
0x0040005c  0000 0000 68b5 0500 0000 0000 68b5 0500  ....h.......h...
0x0040006c  0000 0000 0000 2000 0000 0000 0100 0000  ...... .........
0x0040007c  0600 0000 f09f 0d00 0000 0000 f09f 6d00  ..............m.
0x0040008c  0000 0000 f09f 6d00 0000 0000 0000 0000  ......m.........
0x0040009c  0000 0000 0000 0000 0000 0000 0000 2000  .............. .
0x004000ac  0000 0000 aa7e 1390 efbe adde 8007 0d16  .....~..........
0x004000bc  0000 0000 3404 0d00 3404 0d00 2001 0000  ....4...4... ...
```

Non Unpackable Packer

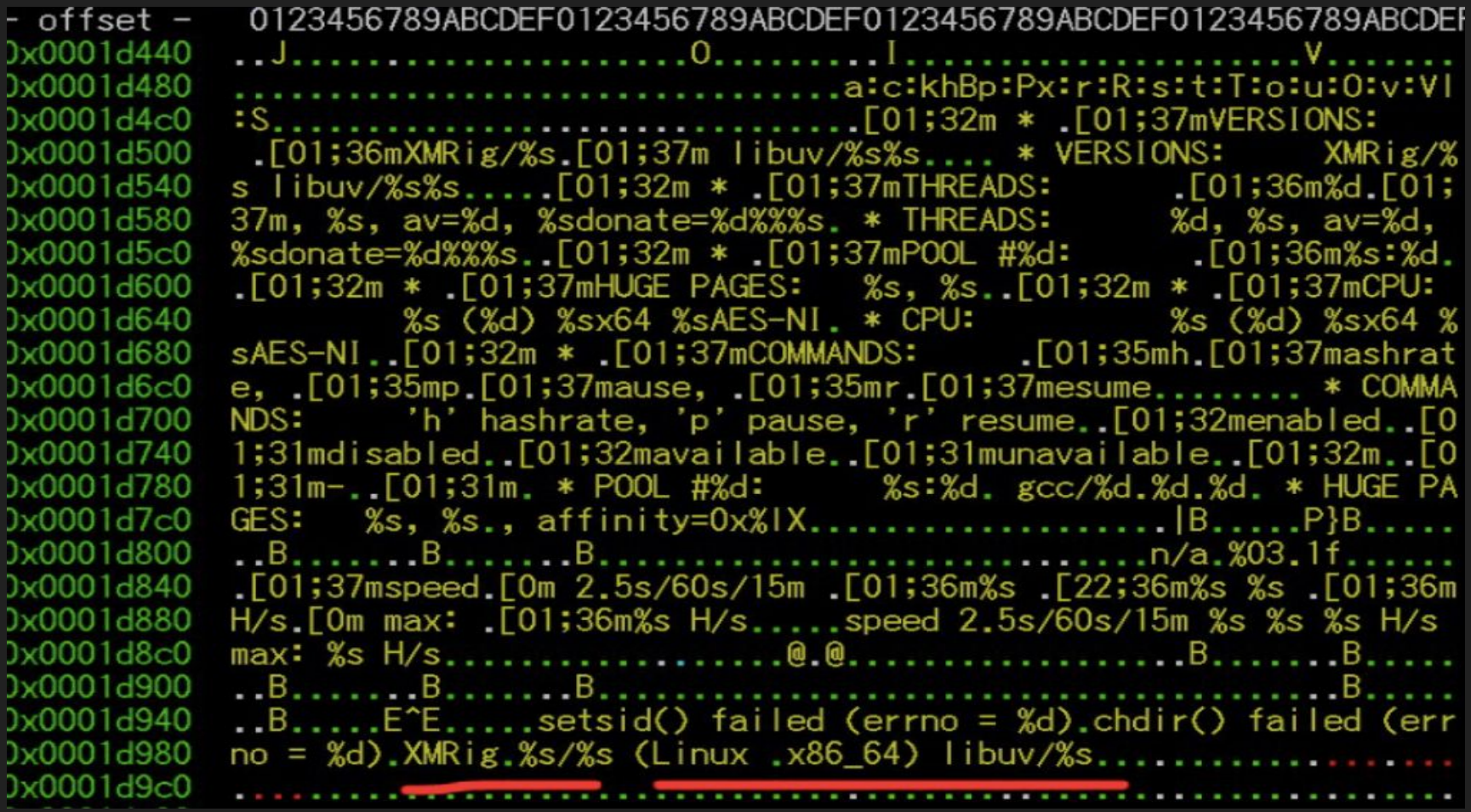
Sometimes you can see the packed's ELF header, BUT it's NOT UPX!!

Non Unpackable Packer

If you fo the debugging.. you can see it is executed in base address

Non Unpackable Packer

Anyway, if you dump it, (command: wf) this one is the XMRig...Unpacked



Non Unpackable Packer

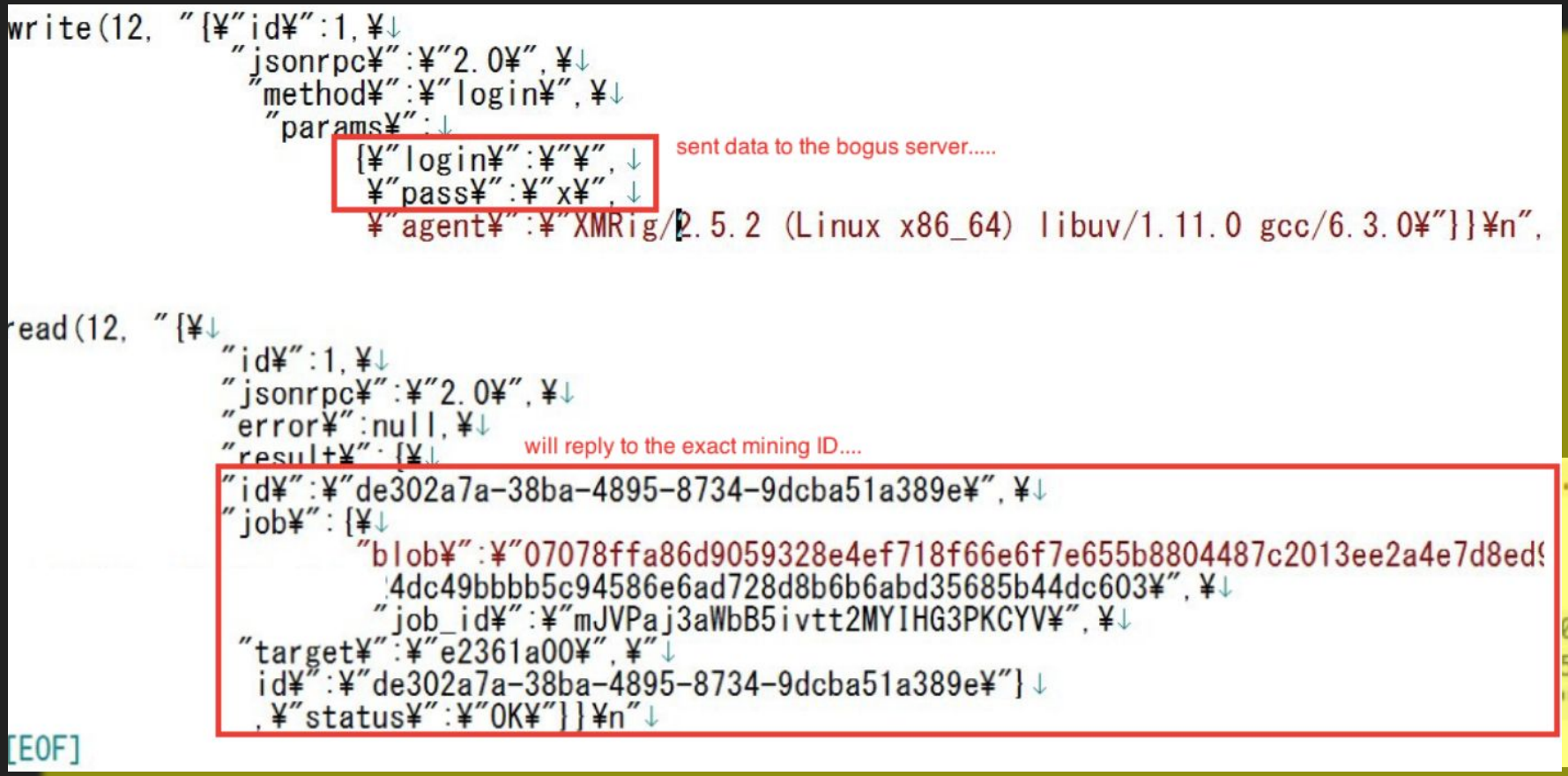
It will run like this without revealing forensics of the real sample is:

```
 * VERSIONS:     XMRig/2.5.2 libuv/1.11.0 gcc/6.3.0
 * HUGE PAGES:   available, disabled
 * CPU:
 * THREADS:      1, cryptonight, av=3, donate=20%
 * POOL #1:      proxy.monoguide.ml:80
 * COMMANDS:     hashrate, pause, resume
2018-06-14 00:55:50] use pool proxy.monoguide.ml:80 5.133.179.36
2018-06-14 00:55:50] new job from proxy.monoguide.ml:80 diff 2500
2018-06-14 00:56:12] dev donate started
2018-06-14 00:56:12] new job from proxy.poolnero.club:80 diff 2500
2018-06-14 00:56:46] speed 2.5s/60s/15m n/a n/a n/a H/s max: n/a H/s
```


```
{"id":1,"jsonrpc":"2.0","method":"login","params":{"login":"","pass":"x","agent":"XMRig/2.5.2
(Linux x86_64) libuv/1.11.0 gcc/6.3.0"}}
{"id":1,"jsonrpc":"2.0","error":null,"result":
{"id":"de302a7a-38ba-4895-8734-9dcba51a389e","job":
{"blob":"07078ffa86d9059328e4ef718f66e6f7e655b8804487c2013ee2a4e7d8ed9a45f654f137838b4f000000
00c23632ca53b467c42c7c24dc49bbbb5c94586e6ad728d8b6b6abd35685b44dc603","job_id":"mJVPaj3aWbB5i
vtt2MYIHG3PKCYV","target":"e2361a00","id":"de302a7a-38ba-4895-8734-9dcba51a389e"},"status":"O
K"}}
```

Non Unpackable Packer

It doesn't use much of anti-debugging, so you can enjoy this luxury:



Why they abuse these miners?

As per stated in the source code of this packed XMRig, looks like the same as Tiny XMR mooner they abused the donation functionality.. and this seems the one of aspects that bad actor(s) would like to hide.

```
42   * If you plan on changing this setting to 0 please consider making a one off donation to my wallet:
43   * XMR: 48edfHu7V9Z84YzzMa6fUueoELZ9ZRXq9VetWzYGzKt52XU5xvqgzYnDK9URnRoJMk1j8nLwEVsaSWJ4fhdUyZijBGUicoD
44   * BTC: 1P7ujsXeX7GxQwHNnJsRMgAdNkFZmNVqJT
45   */
46  constexpr const int kDefaultDonateLevel = 5;
47  constexpr const int kMinimumDonateLevel = 1;
48
49
50  #endif /* __DONATE_H__ */
```

Why they abused these miners?

No matter what, it will mine with actor's pre-set ID anyway..

Donation function is abused as a pre-set hardcoded. Tested! (below)

NUP sample for you

1041ae74caf56d503fabcd411364076a

Conclusion

1. Let’s learn together on how ELF packers evolves
2. ...with don’t forget the BASIC of executable in ELF itself
3. Packers will always be there to protect binaries,
   so...let's ENJOY REVERSING THEM!
4. Using radare2 is always fun. Remember this great motto always:

JUST FIRE R2 AND EVERYTHING IS GOING TO BE OKAY! :)

My recommended reference

1. https://github.com/torvalds/linux/blob/v4.11/fs/binfmt_elf.c
2. http://www.sco.com/developers/gabi/2003-12-17/contents.html
3. http://phrack.org/issues/58/5.htm
4. http://www.skyfree.org/linux/references/ELF_Format.pdf

Others reading maybe can help you in practical level:
http://www.cirosantilli.com/elf-hello-world/
https://medium.com/@MrJamesFisher/understanding-the-elf-4bd60daac571

Good Tools:

1. ElfDump
   https://github.com/spurious/elftoolchain-mirror/blob/master/elfdump/elfdump.c
2. ElfToolChain https://sourceforge.net/projects/elftoolchain
3. PyElfTools https://github.com/eliben/pyelftools

Fin

Let's thank to pancake, radare2 dev, r2con2018 folks, you guys rock!

For the audience, if you find this useful, please:

0x0 - Blog your own found/made packer or share on how-to dissect them
0x1 - Tweet about it
0x2 - Don't forget to mention: @radareorg and hashtags #radare2 + #r2con2018!
0x3 - Present it in the r2con2019!

Question(s)?